VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozšíření projektu GLIPS Graffiti

Extension of GLIPS Graffiti

2016 Bc. Radek Frydryšek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Radek Frydryšek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rozšíření projektu GLIPS Graffiti
Extension of GLIPS Graffiti

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je rozšířit existující otevřený software GLIPS Graffiti (http://glipssvgeditor.sourceforge.net) umožňující jednoduché vytváření základních obrázků ve formátu SVG o funkcionalitu podporující vytváření složitějších kreseb a animaci jednotlivých grafických objektů.

Rozšíření softwaru umožní:
1. Vytváření a vkládání vlastních předdefinovaných tvarů.
2. Vkládání animací jednotlivých objektů.
3. Zobrazování časové osy.
4. Export animace do video streamu.

Práce bude obsahovat:
1. Analýzu a popis současného stavu softwaru GLIPS Graffiti a použitých knihoven.
2. Implementaci uvedených funkcionalit.
3. Popis vytvořených rozšíření s využitím jazyka UML.

Seznam doporučené odborné literatury:

[1] Kolesnikov, A. (2007), Java Drawing with Apache Batik: A Tutorial, BrainySoftware.com.
[2] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. & Makovec, P. (2003), Návrh programů pomoci vzorů: stavební kameny objektově orientovaných programů, Grada Publishing.
Dále podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 25. dubna 2016 .....

Rád bych na tomto místě poděkoval Ing. Davidu Ježkovi, Ph.D. za konzultace a cenné připomínky spojené s vypracováním této diplomové práce.

Abstrakt

Cílem této diplomové práce je rozšířit vektorový editor GLIPS Graffiti o možnosti animací, tvorbu nových tvarů a export do video streamu. V první části práce je popsán editor GLIPS Graffiti a jeho alternativy. Další kapitola obsahuje analýzu zdrojových kódů tohoto editoru. Dále práce popisuje SVG formát a knihovnu Apache Batik. V obou případech je analýza zaměřena na animační možnosti a jejich využití na příkladech. Poslední část této práce tvoří popis implementace jednotlivých funkcí.

Klíčová slova: SVG, GLIPS Graffiti, Apache Batik, Java, Animace

Abstract

Goal of this diploma thesis is extension of vector editor GLIPS Graffiti to support animations, custom shapes and export to video stream. First part of this thesis is description of GLIPS Graffiti and its alternatives. Next chapter contains analysis of source codes from this editor. This thesis also describes SVG format and Apache Batik framework. In both cases is analysis aimed to animations capabilities and its appplication in practice examples. Last chapter of this thesis describes implementation of all new functionalities.

Key Words: SVG, Glips Graffiti, Apache Batik, Java, Animations

Obsah

Seznam použitých zkratek a symbolů

SVG	–	Scalable Vector Graphics
XML	–	Extensible Markup Language
SMIL	–	Synchronized Multimedia Integration Language
SCADA	–	Supervisory Control and Data Acquisition
HMI	–	Human Machine Interface
DMO	–	Document Object Model
API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
GVT	–	Graphics Vector Toolkit
UML	–	Unified Modeling Language
HTML	–	HyperText Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této práce je analýza animačních možností knihovny Apache Batik a SVG formátu. Poznatky z analýzy budou následně využity při rozšíření podpory animací ve vektorovém editoru GLIPS Graffiti.

První část práce je zaměřena především na editor GLIPS Graffiti. V této kapitole bude popsána jeho základní funkcionalita, stav vývoje, četnost verzí a jejich vzájemné rozdíly.

Kromě editoru GLIPS Graffiti budou také v první kapitole uvedeny alternativní editory, které by mohly být teoretickými kandidáty na podobné rozšíření funkcí. Editory jsou krátce představeny a porovnány jejich funkce s GLIPS Graffiti. Ke konci pak budou zmíněny důvody, proč je výše uvedený editor nejvhodnější volbou pro tuto práci.

Následující kapitola bude zaměřena na popis zdrojového kódu GLIPS Graffiti. Kromě grafického jazyka UML bude využit i slovní popis. Vzhledem k rozsáhlosti zdrojových kódů není možné popsat vše, bude tedy kladen důraz na nejdůležitější komponenty, který budou použity při samotné implementaci.

Další kapitola obsahuje popis SVG formátu. Cílem je zjistit jak lze vytvářet nové objekty, jejich animování a zda jsou tyto funkce kompatibilní s uvedeným editorem. Specifikace formátu SVG je velmi obsáhlá, což je důvodem proč jsou v této kapitole zmíněny pouze základy, které jsou nezbytné pro animování objektů.

Následovat bude popis knihovny Apache Batik. Stejně jako je rozsáhlý formát SVG, tak i knihovna Apache Batik umožňuje spoustu funkcí. Ze stejných důvodů jako v předchozí kapitole budou uvedeny pouze jednotlivé moduly knihovny s krátkým popisem. Ke knihovně Apache Batik není příliš mnoho literatury a příkladů implementace. Z tohoto důvodu budou také uvedeny jednotlivé základní operace, pomocí kterých lze vytvářet a ovládat animace.

Následující kapitola bude již zaměřena na samotnou implementaci nových funkcí v editoru GLIPS Graffiti. Kapitola bude rozdělena podle jednotlivých úkolů v zadání. Každý úkol bude obsahovat popis stávajícího stavu, návrh řešení a popis implementace.

Editor pro tvorbu animací je komplexní aplikací a existuje mnoho možností jak jej vylepšit. Ne všechny z těchto vylepšení bylo možné v průběhu implementace uskutečnit. Proto budou v předposlední kapitole uvedeny návrhy na další rozšíření s krátkým popisem možné realizace.

Poslední kapitolou bude závěr, kde budou shrnuty veškeré poznatky získané v průběhu této práce.

2 GLIPS Graffiti

2.1 Popis programu

GLIPS Graffiti Editor je multiplatformový vektorový editor, který pracuje s SVG formátem. Byl vyvíjen francouzskou společností ITRIS. Umožňuje tvorbu jednouché grafiky a je zároveň schopný napojení na SCADA HMI aplikace.

Webová stránka SourceForge, kde je program publikován včetně zdrojových kódů, uvádí poslední aktualizaci z roku 2013[7]. Tento údaj se však týká pouze změny v popisu programu. Poslední změna zdrojových kódů proběhla v roce 2007, kdy byla vydána současná verze editoru. Tato verze má označení 1.5, nicméně většina webových stránek, kde je tento program ke stažení, nabízí předchozí verzi s označením 1.3, která byla vydána v roce 2005.

Verze 1.3 byla distribuována s instalátorem Nullsoft Install System. Ta aktuální je již k dispozici pouze ve formě zip archívu, který obsahuje samostatné spustitelné moduly programu ve formě jar souborů.

2.2 Funkce programu

Editor GLIPS Graffiti zajišťuje takřka veškerou funkcionalitu, kterou nabízí statická verze formátu SVG. V editoru je nabídka základních tvarů typu čtverec a elipsa. Vlastní tvary lze přidávat pomocí nástroje pera, který vytváří tzv. Bézierovu křivku[1], po jejímž uzavření se vytvoří samostatný tvar. Tvary lze seskupovat a tvořit z nich složitější obrazce. Kromě základních geometrických tvarů lze přidávat i textové pole a tzv. JSwing komponenty typu tlačítko, rozbalovací nabídka, apod. Do zobrazené scény lze i naimportovat rastrový obrázek. Veškeré elementy je možné transformovat pomocí posunu jednotlivých bodů nebo změnou úkosu.

Každý z elementů obsahuje vlastnosti definované SVG formátem. Jedná se například o barvu pozadí, typ a barva zvýraznění. Všechny tyto hodnoty lze upravovat v samostatném dialogovém okně.

Výslednou scénu lze exportovat do různých rastrových formátů, jmenovitě se jedná o JPG, PNG, BMP a PDF. Kromě exportu je možný i tisk. Dále je umožněno zobrazení aktuální struktury vytvářeného SVG souboru a vytížení RAM.

2.2.1 SCADA HMI

SCADA je zkratka pro Supervisory Control And Data Acquisition, tzn. Supervizní řízení a sběr dat a HMI označuje Human Machine Interface.[3][21]

SCADA systém je nadřazen HMI. Jedná se o systém, který shromažďuje data z různých čidel a zobrazuje je operátorovi na terminál, kde jsou dále zpracovány. SCADA systémy jsou tvořeny vstupně výstupním hardwarem, regulátory, HMI, softwarem, apod. HMI pak zajišťuje samotné zobrazení informací o stavu různých procesů.[1][20]

[1] Bézierova křivka je typ parametrické křivky, jejíž tvar je definován pomocí řídicích bodů[5]

2.3 Alternativy GLIPS Graffiti

V případě, že se pokusíme najít editor s podobnými parametry jako je GLIPS Graffiti, zjistíme, že mnoho alternativ neexistuje. Níže jsou stručně popsány jednotlivé alternativy, které jsou také vektorovými editory a umožňují export do formátu SVG. Pouze jedna varianta podporuje částečně tvorbu animací. V případě opensource vektorových editorů pracujících s formátem SVG neexistuje téměř žádný, který by zajišťoval plnou podporu animací.

2.3.1 OpenOffice Draw

OpenOffice Draw je vektorový editor, který je přiložen v balíku kancelářských nástrojů OpenOffice[18]. Stejně jako GLIPS Graffiti se jedná opensource. Nabízí poměrně širokou paletu předdefinovaných tvarů. Oproti GLIPS Graffiti zde chybí nástroj pera pro vytváření Bézierových křivek. Jednotlivé tvary stejně jako v GLIPS Graffiti lze měnit za pomocí bodů, kterými jsou tvořeny.

Výchozím formátem bohužel není SVG, ovšem export do tohoto formátu je umožněn. Nicméně výsledný soubor nevyužívá předdefinované elementy z SVG formátu. Nakreslíme-li například obdélník, tak OpenOffice Draw tvar při exportu převede na cestu, namísto aby využil předdefinovaný element rect. Výsledný soubor pak není příliš vhodný pro ruční editaci.

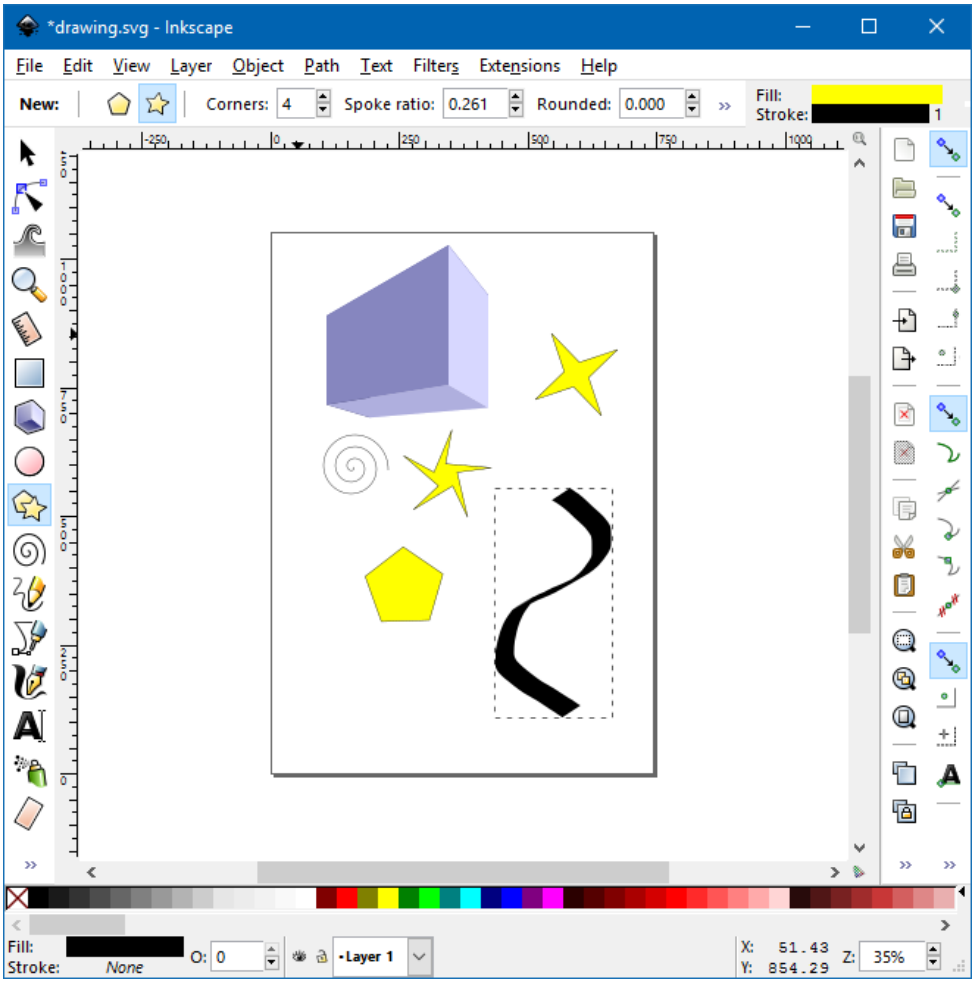
Nabídka podporovaných souborů pro import a export je poměrně široká. Obrázek lze exportovat do celkem 20 formátů, například PNG, PDF, JPEG, BMP, apod.

2.3.2 Inkscape

Inkscape je velice populární multiplatformní opensource vektorový editor. Stejně jako v případě GLIPS Graffiti je nativním formátem SVG. Z pohledu možností tvorby grafiky je tento editor mnohem komplexnější. Je zde především větší nabídka nástrojů, příkladem může být kaligrafické pero anebo větší podpora předdefinovaných tvarů například hvězda, mnohoúhelník, spirála, apod. Také možnosti importu a exportu jsou bohatší. V případě importu je podporováno 60 různých formátů. Export jich pak nabízí 27. Z podporovaných se jedná o běžné formáty typu PNG, JPG, PDF až po méně běžné dxf (AutoCAD DXF R14) nebo sK1 (sK1 Editor).

Inkscape umožňuje přidávat nové funkce prostřednictvím doplňků. Doplňky jsou zpravidla psány v jazycích Python, Perl anebo C++. Jejich tvorba je poměrně dobře popsána na oficiálních stránkách editoru[11].

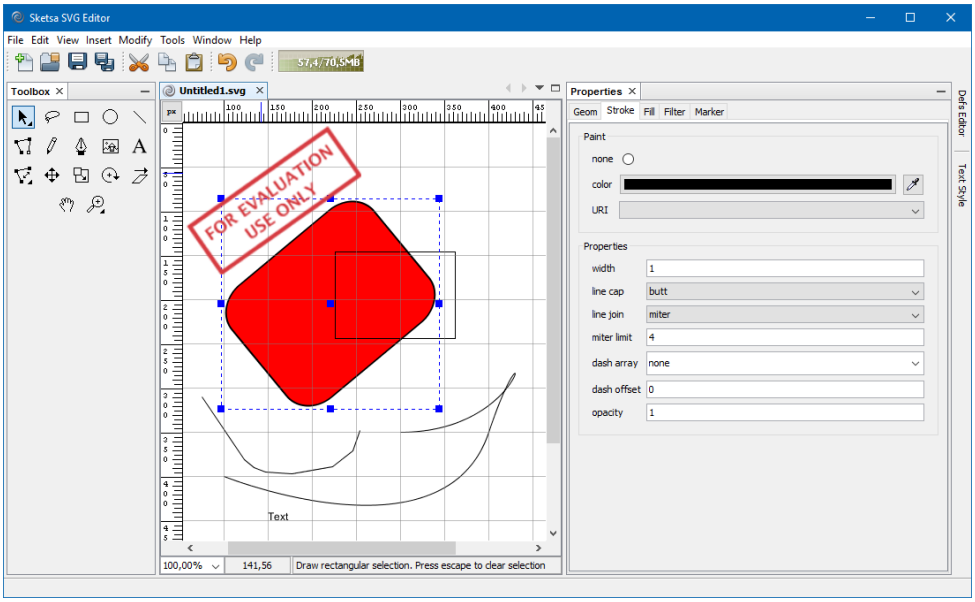
Přestože stávající nabídka doplňků je poměrně bohatá (48), tak ani jeden neumožňuje tvorbu animací.



Obrázek 1: Inkscape editor

2.3.3 Sketsa SVG Editor

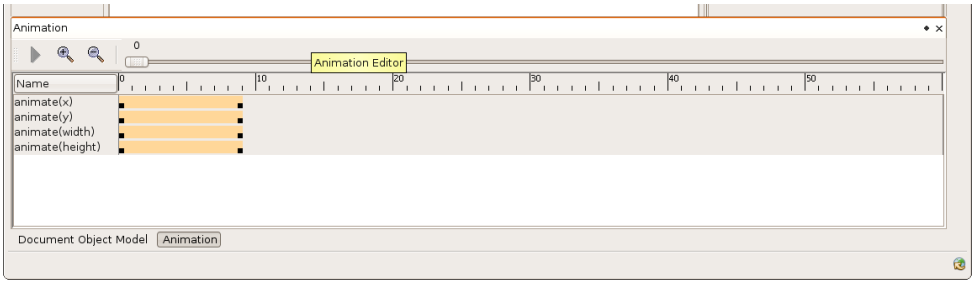
Sketsa stejně jako předchozí editor využívá primárně SVG formát. Také je multiplatformní, ale bohužel se jedná o placený software. Licence pro jednoho uživatele je aktuálně prodávána za 89$. Je také důležité zmínit, že poslední aktualizace programu proběhla v září roku 2013, což může znamenat ukončení vývoje[14]. Co se týče podporovaný funkcí, tak je editor velmi podobný GLIPS Graffiti. Nabízí také pouze základní tvary obdélník a elipsu, nástroj pero, přesun a transformace tvarů, přiblížení. Možnosti exportu jsou oproti GLIPS Graffiti poněkud strohé, export je možný pouze do formátu JPG a PNG. Výhodou v porovnání s GLIPS Graffiti je podpora doplňků. Stávající doplňky nabízí například podporu PDF Exportu, emotikony, knihovnu symbolů a rozšíření předdefinovaných tvarů.



Obrázek 2: Sketsa editor

Na oficiálním blogu společnosti, která zajišťuje vývoj tohoto editoru se lze dočíst o rozpracovaném doplňku zajišťující podporu animací ve formátu SMIL. Nejnovější verze 0.4 je schopna přidávat a editovat animace u jednoho elementu. Nicméně tyto změny se neukládají a pouze demonstrují možnou funkcionalitu. Doplněk je k dispozici ke stažení, je ale nezbytná vývojová verze editoru, která již není dostupná. Představu o doplňku si lze tedy vytvořit pouze z jednoho obrázku (viz. Obrázek 3) a seznamu plánovaných změn.

Vzhledem k tomu, že verze doplňku 0.4 byla vydána v roce 2007 a dosud nebyla aktualizována, lze předpokládat že vývoj byl ukončen.



Obrázek 3: Časová osa v editoru Sketsa

Samozřejmě existuje více editorů, které by vyhovovaly těmto požadavkům, není jich však mnoho. Tyto příklady pouze ilustrují fakt, že GLIPS Graffiti je pro tuto práci nejvýhodnějším řešením. Každá z těchto variant má své výhody a nevýhody. Například OpenOffice Draw je spíše orientovaný na tvorbu diagramů.

Inkscape je naopak velice komplexní nástroj pro tvorbu vektorové grafiky, podpora animací by u něj byla realizovatelná, nicméně z důvodu jeho komplexity by bylo velice obtížné a časově

náročné analyzovat stávající kód. Samozřejmě by bylo možné vytvořit doplněk s podporou animací. Problémem by byl ovšem fakt, že doplňky jsou psány v jazycích Perl, Python nebo C++. Pro tyto jazyky neexistuje knihovna s podporou SVG formátu na takové úrovni jako Apache Batik.

Vhodným adeptem pro rozšíření o podporu animací je Sketsa SVG Editor. Stejně jako GLIPS Graffiti využívá Apache Batik knihovnu, podpora animací je již rozpracovaná a především není editor příliš složitý. Bohužel se však nejedná o opensource, tudíž nelze jej nijak dodatečně upravovat, pouze prostřednictvím doplňků. GLIPS Graffiti z těchto variant je nejlepším kompromisem. Jsou k dispozici zdrojové kódy, editor není příliš komplexní, přesto nabízí dostatečné množství funkcí pro jednoduchou tvorbu grafiky.

3 Popis zdrojových kódů editoru GLIPS Graffiti

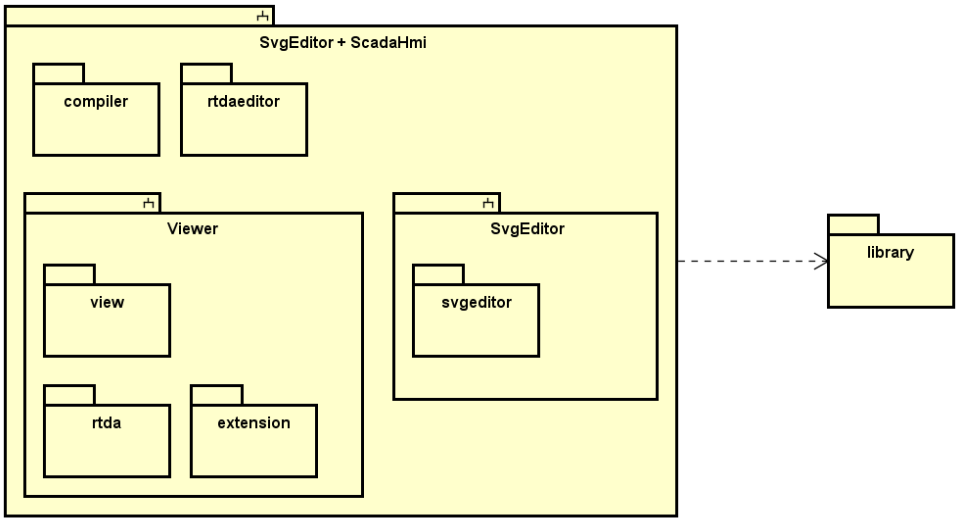
Jak již bylo zmíněno výše, aktuální verze je distribuována formou zip archívu, jehož obsahem jsou základní moduly. Základními moduly se myslí GLIPSGraffiti.jar, GLIPSGraffitiSCADA.jar a GLIPSView.jar. GLIPGraffiti.jar a GLIPSGraffitiSCADA.jar spouští samotný editor. V prvním případě se jedná o odlehčenou verzi bez podpory SCADA, druhý soubor již obsahuje plnohodnotnou verzi. GLIPSView.jar je jednoduchá aplikace, která slouží pro otevření SVG souboru a zobrazení na celou obrazovku.

Zdrojové kódy editoru obsahují základní dva rodičovské balíky GLIPSGraffiti a Libraries. První jmenovaný obsahuje zdrojové kódy editoru a druhý veškeré externí knihovny. Editor využívá Apache Batik verze 1.6, přičemž některé nalezené soubory ve zdrojových kódech odpovídají vývojové verzi 1.7 beta. Další použité knihovny třetích stran jsou JGoodies Looks, PngEncoder a Lowagie iText.

JGoodies Looks je jedna z mnoha knihoven, které poskytuje společnost JGoodies. V tomto případě se jedná o balík grafických komponent, které slouží k přizpůsobení vzhledu aplikace, tedy nastavení tzv. Look and Feel. [2].

PngEncoder a Lowagie.Text slouží pro export do formátu PNG a PDF. V obou případech se jedná o volně dostupné knihovny, rozdíl je pouze v použitých licencích. PngEncoder je šířen s licencí GNU GPL v2 a Lowagie.Text pak využívá MPL/LGPL, přičemž její novější verze iText je distribuována za pomocí AGPL. Pro obě knihovny tedy platí, že jejich použití je podmíněno zveřejněním zdrojových kódů.

Celkovou strukturu projektu nejlépe popisuje diagram, viz. Obrázek 4.



Obrázek 4: Struktura projektu

[2]Vzhled a chování standardních Swing komponent je definován prostřednictvím tzv. Look and Feel objektu, který je reprezentován abstrakní třídou ComponentUI. [10]

Výchozím balíčkem je fr.itris.glips, který obsahuje jednotlivé základní komponenty (stejnojmenné adresáře) view, rtda, extensions, atd. Diagram rozděluje balíčky na funkční celky, které byly zmíněny v prvním odstavci. Každý z těchto celků využívá společný balíček library.

Grafickou část editoru zajišťuje balíček svgeditor, který dále obsahuje celkem 13 dalších adresářů. V případě některých rozsáhlejších komponent jsou tyto adresáře členěny na další balíčky. Jednotlivé komponenty jsou pojmenovány velice přehledně a nemá je smysl dlouze popisovat. Proto je zde uveden pouze výpis jejich názvů a k nim jednoduchý popis.

- actions

  V tomto balíku jsou umístěny moduly DomActionsModule a CliboardModule. První zajišťuje provádění akcí nad DOM strukturou otevřeného souboru. Příkladem může být modul, který zajišťuje rotaci elementů (kolekce objektů třídy Element) RotateModule, který odvozuje DomActionsModule. Druhý jmenovaný modul poskytuje přístup k systémové schránce. Kromě těchto modulů je zde ještě definována nástrojová lišta.

- colorchooser

  Balík colorchooser obsahuje rozšíření JColorChooser komponenty pro podporu SVG barev.

- display

  Kreslící plátno a všechny funkce s ním související jsou definovány v tomto balíčku. Balíček se dále dělí na cavas, handle, selection a undoredo. Jak již názvy napovídají jedná se o definici plátna, plovoucí okno s otevřeným souborem, selekce a undo-redo, tedy podpora pro vrácení provedené akce její novou aplikaci za pomocí klávesových zkratek ctrl+z a ctrl+y.

- io

  Veškerou práci se soubory (otevírání, ukládání, export, apod.) zajišťují třídy obsažené v tomto balíčku. Výchozí třídou je IOManager, která umožňuje přístup k jednotlivým konkrétním implementacím, například FileNew, FileExport, apod.

- options

  Tvorba grafických elementů obsahuje v editoru pět základních módů. Jsou jimi zachování měřítka, vynucení rovných čar pro nástroj pero, deaktivace automatické selekce nových tvarů, automatické uzavírání cest a zarovnání podle pravítek. Tyto módy lze různě kombinovat mezi sebou. Definici jednotlivých položek v menu pro tyto módy zajišťuje třída OptionsModule, která je obsažena ve výše zmíněném balíčku. Správu nastavení pro každý mód zajišťují vlastní třídy pojmenované podle daného módu, například SquareModeManager, RemanentModeManager, apod.

- properties

Každý grafický element v SVG má své vlastnosti, například barva výplně, barva a styl zvýraznění, identifikátor apod. Všechny tyto atributy jsou spravovány v nástroji Properties, jehož definice je obsažena ve stejnojmenném balíku properties.

- resources

  Balík resources obsahuje veškeré externí zdroje, které jsou použity v aplikaci. Externími zdroji se myslí překlady textů (ve výchozím stavu angličtina a francouzština), obrázky a ikony. Dále jsou v tomto balíku umístěny XML soubory, které obsahují seznam modulů, položek v menu a nástrojové liště. Správa zdrojů je pak detailněji popsána v samostatné kapitole 3.2.

- selection

  Tento balík obsahuje pouze dvě třídy a to SelectionModule a SelectionInfoManager. První z nich představuje modul selekce, ve kterém je zajišťována funkcionalita nástrojů pracujících se selekcí (Selekce vybrané zóny, Vybrat vše, Vytvořit skupinu, apod.). SelectionInfoManager pak umožňuje přístup k módu selekce (selekce zóny, standardní selekce kliknutím, selekce zóny s přiblížením, případně deaktivace selekce).

- shape

  V tomto balíku se nalézají veškeré třídy, které umožňují práci se základními tvary v editoru. Důležité jsou především AbstractShape, RectangularShape a PathShape. První je abstraktní třída, kterou rozšiřují všechny další třídy představující tvar, případně s vyšší úrovní abstrakce, například RectangleShape představující obdélník rozšiřuje RectangularShape, která pak dědí z AbstractShape. PathShape představuje tvar vytvořený z cesty (křivky).

- undoredo

  GLIPS Graffiti podporuje funkci pro vrácení provedeného kroku a jeho znovuprovedení, tedy "undo redo". Balíček obsahuje třídu, která zajišťuje grafickou reprezentaci této funkcionality. Zobrazuje položky v menu pro krok zpět, a znovu s příslušnými popisky, které udávají popis daného kroku.

- visualresources

  V SVG formátu je možné definovat vlastní přechody, značky a vzorce. Tento balík obaluje komponentu pro jejich vytváření. Samotná komponenta (lépe modul, viz. další kapitola) je reprezentována třídou SVGVisualResources. V tomto balíku se dále vyskytují třídy s tímto "prefixem"v názvu. Tyto třídy pak představují komponenty v rozhraní tohoto modulu - příkladem je rozbalovací nabídka, její prvky, apod.

- widgets

Jednoduché komponenty rozšiřující abstraktní třídu Widget, která odvozuje JComponent. V případě balíku svgeditor je zde pouze jedna třída ColorChooserWidget, která představuje rozbalovací nabídku pro výběr barvy, ta je použita například pro nastavení barvy pomocné mřížky. Další komponenty jsou obsaženy v balíku libraries, typicky se jedná o podobně jednoduché třídy.

Balíček fr.itris.glips obsahuje kromě výše zmíněných komponent také hlavní výchozí třídu EditorMain, jejíž součástí je statická metoda main, která spouští editor. Samotnému spuštění předchází vytvoření okna editoru a následná inicializace hlavních modulů, viz. následující kapitola 3.1.

3.1 Modulární systém, stávající moduly a přidání nového modulu

Vektorový editor GLIPS Graffiti je tvořen moduly, které zajišťují přístup k základní funkcionalitě. Cílem každého modulu je vytvořit tlačítka (v menu nebo nástrojové liště) a zaregistrovat reakci na událost pro kliknutí pro každou z dostupných funkcí editoru. Samotná funkcionalita je pak rozložena do dalších tříd.

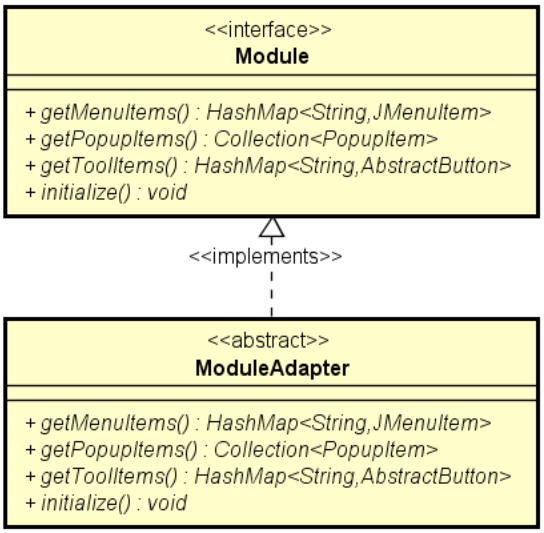
Seznam všech modulů a cest k jejich definicím (třídám) je uložen v souboru modules.xml. Každý záznam je tvořen cestou k definici (viz. Výpis 1).

```
<?xml version="1.0" ?>
<modules>
    <module class="fr.itris.glips.svgeditor.io.module.IOModule"/>

    <module class="fr.itris.glips.svgeditor.MemoryMonitorModule"/>
    <module class="fr.itris.glips.svgeditor.actions.clipboard.ClipboardModule"
        />
    <module class="fr.itris.glips.svgeditor.undoredo.UndoRedoModule"/>
    <module class="fr.itris.glips.svgeditor.options.OptionsModule"/>
</modules>
```

Výpis 1: Seznam modulů

Při startu aplikace je načten tento xml soubor a v cyklu se pak prochází každý modul. Pomocí reflexe je modul inicializován, přičemž se počítá s parametrickým konstruktorem obsahující argument třídy Editor. Pakliže inicializace selže, například není nalezen konstruktor s parametrem Editor, tak nedochází k přerušení startu aplikace, ale pouze k přeskočení načtení daného modulu.

Každý modul musí implementovat rozhraní Module (diagram viz. Obrázek 5). Toto rozhraní obsahuje čtyři základní metody, zajišťující základní funkcionalitu modulu.



Obrázek 5: Rozhraní Module

getMenuItems

Cílem této metody je inicializace všech položek menu daného modulu. Návratovou hodnotou je kolekce typu HashMap<String, JMenuItem>, kde každý prvek reprezentuje identifikátor a položku v menu. Klíče jednotlivých záznamů kolekce odpovídají definici nabídky menu v XML souboru menu.xml.

getPopupItems

Návratová hodnota této kolekce je kolekce Collection<PopupItem>. Prvek kolekce, představuje položku v nabídce, která se zobrazí po kliknutí pravým tlačítkem na objekt. Struktura nabídky je definována v popup.xml.

getToolItems

Metoda getToolItems je velmi podobná první metodě getMenuItems. I zde je použita kolekce typu HashMap<String, AbstractButton>. Princip je totožný, klíčem je řetězec definovaný v souboru tool.xml, hodnotou je objekt typu AbstractButton představující tlačítko v nástrojové liště.

initialize

Metoda initialize slouží k dodatečné inicializaci modulu. K volání této metody dochází až po úspěšném vytvoření objektu modulu, tedy zavolání jeho konstruktoru.

Všechny tyto metody jsou veřejné a jsou volány v průběhu spouštění programu, kdy se inicializují veškeré moduly. Ne každý modul však musí nezbytně implementovat všechny tyto metody. Může být například modul, který nemá žádnou položku v kontextové nabídce vybraného elementu, tedy není nutné implementovat metodu getPopupItems. Pro tyto případy slouží třída ModuleAdapter, která implementuje rozhraní Module. Všechny metody, mající návratovou

hodnotu zde vrací null. Implementace těchto metod je pak provedena pomocí překrytí (anotace override).

3.1.1 Moduly

Glips Graffiti obsahuje celkem 28 modulů. Každý z nich zajišťuje sobě určenou funkcionalitu. Ne všechny jsou však nezbytné pro práci s animací. Z tohoto důvodu budou popsány pouze ty, které byly využity při implementaci nové funkcionality, případně jde o moduly zajišťující základní funkce editoru typu export, selekce, otevření souboru apod.

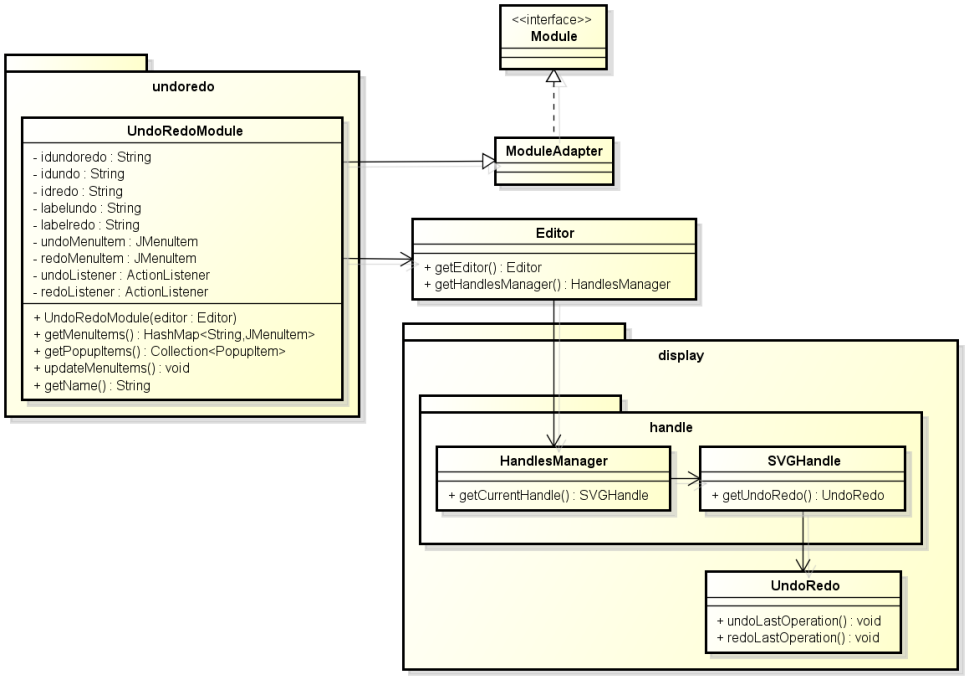
IOModule

Module IOModule zajišťuje vytvoření všech položek menu a v nástrojové liště, které zajišťují práci se soubory. Vytvořením je myšlena inicializace tlačítek, nastavení jejich popisků dle lokalizace a napojení akcí na událost po kliknutí na tlačítko. Samotná funkcionalita jednotlivých částí je pak řešena v samostatných třídách z balíčku io.managers.

UndoRedoModule

Tento modul umožňuje funkci pro krok zpět a vpřed. Stejně jako u téměř většiny ostatních modulů se v této třídě vytváří položky v menu a napojují se jejich události na konkrétní akce, samotná funkcionalita je pak zajištěna v dílčích třídách. Funkce undo a redo jsou navázány na jednotlivé akce v rámci otevřeného souboru. Funkcionalita je implementována ve třídě UndoRedo, ke které přistupuje samotný SVGHandle představující aktuální soubor SVG.

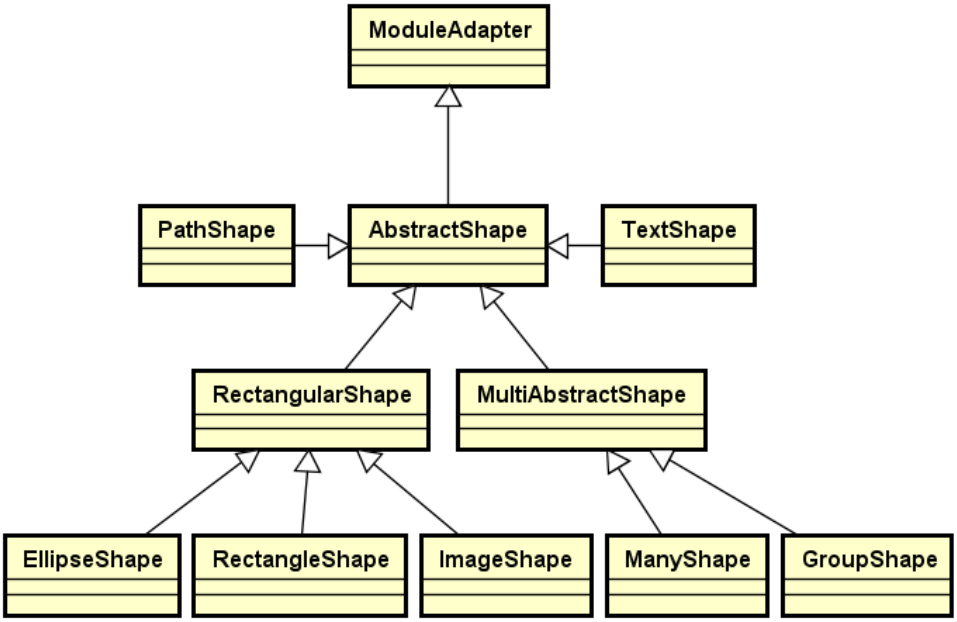
Implementace undo, redo je vyřešena pomocí kolekce instancí třídy Runnable, kde každý prvek představuje akci, která bude provedena v novém vlákně. V případě přidání nové akce se vždy musí vložit jak operace s provedením akce, tak i její opačná varianta, která vrátí soubor do předchozího stavu. Níže je zobrazen diagram implementace modulu (viz. Obrázek 6), bohužel z prostorových důvodů musí být zmenšen. V příloze je uvedena jeho plnohodnotná verze ve vyšším rozlišení (viz. Obrázek 26).



Obrázek 6: Modul UndoRedoModule

RectangleShape, EllipseShape, PathShape

Každý z obrazců, který lze v Glips Graffiti vytvářet má vlastní modul. Pro elipsu je definována třída EllipseShape, křivku PathShape, apod. Tyto moduly rozšiřují abstraktní třídu AbstractShape, kde jsou definovány společné operace pro všechny obrazce. Architekturu jednotlivých tvarů zobrazuje zjednodušený třídní diagram níže (viz. Obrázek 7).



Obrázek 7: Architektura tvarů v editoru

SelectionModule

SelectionModule zajišťuje inicializaci všech položek v menu a nástrojové liště, které souvisí se selekcí elementů. Jedná se o funkcionalitu typu vybrat vše, odznačit vše, vytvořit skupinu elementů, apod. Shlukování elementů do skupin je řešeno přímo v třídě modulu, ostatní funkce selekce jsou implementovány ve třídě SVGHandle, která má přístup k vybraným elementům daného plátna.

GridModule

Na kreslící plátno v editoru lze zobrazit pomocnou mřížku. U této mřížky lze nastavit styl čáry (plná, tečkovaná, čerchovaná apod.), barvu a tloušťku. Zároveň lze nastavit i jednotlivé rozestupy přímek.

Modul se skládá ze čtyř tříd. Samotný modul tvoří třída GridModule, která definuje položky v menu, nástrojové liště, popisky a skrývání ovládacích prvků pokud není otevřen žádný soubor. Další třídou je Grid, která zajišťuje samotné vykreslování pomocných čar. Mřížka je vykreslována pomocí Graphics2D API. Dialog pro nastavení parametrů je definován v třídě GridParametersDialog. Veškeré nastavení je pak vnořeno do třídy GridParameters.

SVGProperties a SVGVisualResources

Modul SVGProperties zajišťuje zobrazení vlastností elementu a jejich úpravu. Druhým modulem je SVGVisualResources, pomocí kterého, je možné vytvářet barevné přechody, vzorky a značky, které lze aplikovat na zobrazené objekty.

3.1.2 Správa nastavení modulů

Přímý přístup k modulům je umožněn pomocí třídy ModuleManager, která si uchovává kolekci objektů typu Module. Případně je tato třída schopna za pomocí reflexe inicializovat na základě souboru modules.xml nový modul.

Nastavení modulů a veřejný přístup je pak zpravidla umožněn prostřednictvím tříd se sufixem Manager. Například ClipboardManager pro modul ClipboardModule. Níže je uveden seznam několika tříd, které spravují základní moduly.

ResourcesManager

ResourceManager je třída, která slouží pro správu všech externích zdrojů použitých v aplikaci. Těmito zdroji se rozumí především ikony a texty. V aplikaci se pro sestavení položek v menu, seznam modulů používají XML soubory. Dále je zde, zajištěna podpora pro ukládání nedávných souborů do registrů systému.

IOManager

Tato třída zajišťuje přístup ke všem implementacím, které řídí vstupně výstupní operace aplikace. Příkladem může být otevření souboru, uložení souboru, export, nový soubor apod.

ColorChooser

Třída rozšiřující standardní Swing komponentu pro výběr barvy JColorChooser. Komponenta je rozšířena o podporu barev W3C standardu a jejich identifikátory.

HandlesManager

V editoru je každý otevřený soubor zobrazen v plovoucím okně, které obsahuje plátno pro editaci. Každé taková otevřená scéna je v Glips Grafiti reprezentována třídou SVGHandle a právě třída HandlesManager zajišťuje přístup ke všem instancím této třídy. Dále obsahuje základní metody pro získání počtu otevřených souborů, jejich aktivaci, případně deaktivaci apod.

ClipboardManager

ClipboardManager zajišťuje přístup k systémové schránce. Udržuje kolekci elementů z aktuálně otevřeného SVG souboru, které byly přidány do schránky pomocí klávesové kombinace ctrl+c.

ScquareModeManager

Kreslení pomocí pera má několik módů a nastavení. Pro každé nastavení je zavedena v systému samostatná třída. První z nich je SquareModeManager, která zajišťuje přístup k nastavení zachování poměru stran při změně velikostí objektů.

RemanentModeManager

Standardní chování v editoru je, že po dokončení tvorby cesty nebo tvaru se ukončí režim kreslení a je automaticky zvolena selekce daného objektu. Zapnutím tohoto módu zůstane aktivní mód kreslení a lze dále pokračovat jiným tvarem.

ClosePathModeManager

Pakliže uživatel vytváří Bézierovou křivku za pomocí pera, tak její uzavření a následná konverze na nový tvar musí být provedena manuálně vybráním volby pro uzavření cesty. V případě zapnutí tohoto módu jsou vytvářené křivky uzavírány automaticky.

ConstraintLineModeManager

Tento mód omezuje koncové body pro vytváření linek. Jedná se o mód, který usnadňuje kreslení přímek, přičemž úhel mezi předchozím a následujícím bodem je omezen na kroky po 90°.

Všechny čtyři výše popsané moduly zajišťují pouze správu aktuálního nastavení daného módu. Jsou převážně tvořeny metodami pro získání aktuální hodnoty nastavení formou logické hodnoty a její aktualizace. Samotná funkcionalita je pak zajištěna v modulech jednotlivých tvarů.

ModuleManager

Tato třída spravuje přístup k jednotlivým modulům, které zajišťují funkcionalitu celého editoru.

3.2 Lokalizace a správa prostředků

Lokalizace a obecně celá správa prostředků v aplikaci je řešena pomocí abstraktní třídy ResourceBundle z balíčku java.util[4]. Logiku správy prostředků zapouzdřuje třída ResourcesManager, která obsahuje statickou instanci třídy ResourceBundle. Pro získání lokalizovaného textu, stačí použít tuto instanci a její metodu getString, kde do parametru vložíme identifikátor textu. Pro ikony je pak připravena ve třídě ResourcesManager veřejná statická metoda getIcon, která vrací objekt třídy ImageIcon. Kromě správy textů a ikon obsahuje třída ResourceManager také metody pro přidávání a odebírání naposledy otevřených souborů.

3.3 GUI modulu

Moduly, které obsahují uživatelské rozhraní využívají třídu ToolsFrame. Tato třída, zajišťuje funkcionalitu pro zobrazení, resp. skrytí okna. Oknem je zde myšleno plovoucí dialogové okno (třída JInternalFrame) jehož nadřazeným prvkem je samotný editor. Změna viditelnosti okna je navázána na položku v menu a nástrojové liště. Kromě metod pro zjištění id modulu, získání položky v menu a podobných je k dispozici také veřejná metoda, jež umožňuje spuštění paralelní akce při změně viditelnosti okna. Příkladem využití této metody může být inicializace seznamu animací v případě zobrazení okna.

3.4 Správa nastavení aplikace

Veškeré nastavení aplikace se ukládá do registrů systému. K tomuto účelu slouží Java References API.

Nastavení se ve výchozím stavu ukládá do větve `HKEY_CURRENT_USER\Software\JavaSoft\Prefs`. V případě GLIPS Graffiti je celá cesta `HKEY_CURRENT_USER\Software\JavaSoft\Prefs\`

`fr\itris\glips`. V unixových systémech se nastavení ukládá do domovského adresáře uživatele a následně do `.java/.userPrefs`.

Java Preferences API nabízí mnoho metod pro správu nastavení aplikace. Nicméně v editoru je nezbytná funkcionalita pouze pro přidání nového nastavení a jeho čtení. Z důvodu zjednodušení práce je třída java.util.prefs.Preferences obalena třídou PreferencesStore, která přesměrovává tuto funkcionalitu prostřednictvím základních metod.

4 Formát SVG

SVG formát, tedy Scalable Vector Graphics je značkovací jazyk popisující dvoudimenzionální vektorovou grafiku, včetně kombinace s rasterovou grafikou v XML. Formát je vyvíjen od roku 1998, přičemž první verze byla publikována W3C konsorciem v roce 2001. Roku 2003 byla vydána verze 1.1, kde byla zásadně vylepšena práce s textem. Specifikace formátu SVG začala být poměrně komplikovaná a od této verze je členěn do subverzí. Plnohodnotná verze je označována jako SVG Full popřípadě celým názvem SVG 1.1 2nd Edition.

SVG Basic

SVG Basic je odlehčený standard, který je primárně určen pro mobilní zařízení. V této verzi chybí nebo je pouze částečně podporována filtrace a použití ořezových cest.

SVG Tiny

SVG Tiny je základním standardem, který obsahuje pouze nezbytné součásti plnohodnotného SVG. Chybí zde podpora skriptů, filtrů, přechodů barev, vzorků, průhlednost a především podpora CSS stylování.

4.1 Základní tvary v SVG

Jak již bylo zmíněno výše, SVG formát je schopen vykreslovat vektorovou grafiku. Mezi základní podporované tvary patří obdélník, kruh, elipsa, čára, tzv. polyline a polygon[12].

4.1.1 rect

Pomocí elementu rect lze vytvořit obdélník o různé šířce, výšce, je možné nadefinovat typ obrysu, jeho barvu, zakulacené rohy, apod. Příklad tohoto elementu je zobrazen viz. Výpis 2.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" width="640" height="480">
<rect x="0" y="0" height="100" width="200" stroke="red" fill="blue" stroke-
    width="5" />
</svg>
```

Výpis 2: Obdelník v SVG

Pozici obdélníku nadefinujeme pomocí atributů x a y. V případě tohoto elementu se pozice počítá od levé horní hrany a váže se k rodičovskému elementu (nejbližší nadřazený element). V našem případě tedy element svg. Velikost elementu lze ovlivnit pomocí atributů height (výška) a width (šířka).

Styly objektů lze ovlivňovat formou kaskádových styků, zde je použita barva výplně a obrysu. Lze tedy přímo do elementu zapsat jednotlivé atributy, případně sjednotit do atributu style anebo nalinkovat externí soubor s CSS styly.

4.1.2 circle

V případě kružnice je pozice definována pomocí dvojice atributů cx a cy. Na rozdíl od obdélníku, zde tyto souřadnice uvádějí vzdálenost mezi středem kružnice a rodičovským elementem. Dalším povinným atributem je poloměr kružnice, který se udává pomocí atributu r.

4.1.3 ellipse

Pozicování elipsy je shodné s kružnicí, jediným rozdílem je definice rozměru. Je nutné uvést poloměr pro obě osy, tedy musí být vyplněny atributy rx a ry.

4.1.4 line

Jednoduchou linku lze zobrazit pomocí elementu line. Pozice, směr a délka je definována za pomocí atributů x1, y1, x2 a y2. První dva slouží pro definici počátečního bodu a zbylé pro koncový bod.

4.1.5 polyline

Jednotlivé linky lze spojit a vytvořit z nich vlastní tvar. Toho lze docílit pomocí elementu polyline, viz. Výpis 3.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" width="640" height="480">
<polyline points="0,0 30,0 15,30"/>
</svg>
```

Výpis 3: Polyline element

Tvar je definován pomocí kolekce bodů v atributu points. Zápis na příkladu výše zobrazí rovnostranný trojúhelník.

4.1.6 polygon

Element polygon je shodný s elementem polyline. Jediný rozdíl je v tom, že polygon automaticky uzavírá cestu definovanou kolekcí bodů. Vzhledem k této vlastnosti je polygon použitelný pro objekt o minimálně třech bodech, tedy případ kdy lze uzavřít cestu.

4.1.7 path

Nejdůležitějším a nejsložitějším elementem je cesta. Pomocí elementu path lze nakreslit Bézierovu křivku, případně cestu uzavřít a tímto vytvořit nový tvar.

Cesta je stejně jako v předchozích případech tvořena kolekcí bodů. Tyto body se zapisují jako hodnoty do atributu d. Ukázka definice cesty, viz. Výpis 4 níže.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" width="640" height="480">
<path d="M50,50 A30,50 0 0,1 100,100" />
</svg>
```

Výpis 4: Path element

Příklad výše zobrazuje jednoduchou křivku. Počáteční bod je uvozen písmenem M, které znamená posun pera na danou pozici bez kreslení. Následující bod obsahuje písmeno A, které značí křivku (Arc).

Tento příklad ukazuje pouze základ. Pomocí elementu path lze kreslit velmi složité tvary s využitím mnoha nastavení.

4.2 Animace

Formát SVG nabízí tři různé formy animování objektů.

- Animování pomocí skriptu na straně klienta
- Animace prostřednictvím CSS animací
- Deklarativní definice

Nejkomplexnější způsob je jednoznačně skriptování na straně klienta. Kód lze psát pomocí jazyka JavaScript, včetně knihovny Jquery. Tímto způsobem lze vytvořit animace, které reagují na akce klienta, například po kliknutí tlačítka, nebo tažení kurzoru. Jednoznačnou výhodou je kompatibilita napříč webovými prohlížeči.

Další možností je vytvářet animace pomocí definice CSS stylu. Tato metoda je pro použití v této práci poměrně nevhodná. Ne všechny prohlížeče ji podporují a především, stejně jako u předchozí metody není dobrá její podpora v použité knihovně Apache Batik.

Poslední možností je tvorba animací formou deklarativní definice ve formátu SMIL.[3] Animace se vytváří pomocí XML elementů. Parametry animací jsou definovány v atributech elementů.

[3]SMIL, neboli Synchronized Multimedia Integration Language je značkovací jazyk vycházející z XML pro vytváření multimediálního obsahu. [22]

Cílem animace je vždy rodičovský element. Tato forma animací je velmi přehledná a hlavně je plně podporována v knihovně Apache Batik.

SVG nabízí mnoho možností jak animovat objekty. V této kapitole budou krátce popsány všechny elementy, které zajišťují deklarativní animování objektů. Kromě jednoduchých animací lze vytvářet i pokročilé animace, které na sebe navzájem navazují.

4.2.1 animate

Element animate slouží k změně jednoho atributu v průběhu času. Ukázka změny viditelnosti pomocí elementu animate, viz. Výpis 5 níže.

```
<rect>
  <animate attributeType="CSS" attributeName="opacity"
         from="1" to="0" dur="5s" repeatCount="indefinite" />
</rect>
```

Výpis 5: Animace změna viditelnosti

U všech elementů animací je nezbytné, aby byly vnořené do elementu, který animují. V našem případě jde o element rect, tedy obdélník.

Jak již bylo zmíněno výše, element animate mění určitý atribut. V tomto případě se jedná o atribut opacity (průhlednost) přičemž hodnota se bude pohybovat od 1 do 0, tedy ze 100% průhlednosti do 0%.

Každý element by měl mít definován celkový čas a počet opakování. Zde je počet opakování neomezený a celkový čas 5 sekund. SVG formát umožňuje udávat jednotky v milisekundách (ms), sekundách (s), minutách (min) a hodinách (h). Kromě celkového času lze specifikovat i počáteční čas pomocí atributu begin. Pokud není tento atribut uveden, tak animace začíná ihned.

4.2.2 set

Element set je zjednodušenou variantou animate. Pomocí tohoto elementu lze v určitém čase nastavit hodnotu vybranému atributu. Na rozdíl od animate nelze nastavit opakování a tedy tato akce se provede pouze jednou. Ukázka použití elementu set, viz. Výpis 6 níže.

```
<circle cx="30" cy="30" r="25" style="stroke: none; fill: #0000ff;">
    <set attributeName="r" attributeType="XML"
         to="100"
         begin="3s" />

</circle>
```

Výpis 6: Změna poloměru kružnice

Příklad výše nastaví po třech sekundách atribut r, tedy poloměr kružnice na 100 místo 25.

4.2.3 animateMotion

AnimateMotion je element, který zajišťuje komplexnější pohyb objektu. Pohyb je dán křivkou, tedy stejným zápisem jako u elementu path, viz. 4.1.7 Path. Animace pohybu po křivce je zobrazena v ukázce níže, viz. Výpis 7.

```
<rect x="0" y="0" width="30" height="15"
       style="stroke: #ff0000; fill: none;">
     <animateMotion
            path="M10,50 q60,50 100,0 q60,-50 100,0"
            begin="0s" dur="10s" repeatCount="indefinite"
            />
  </rect>
```

Výpis 7: Pohyb po křivce

Je důležité podotknout, že pozice animovaného elementu nyní nejsou závislé na rodičovském objektu, ale na křivce, po které se objekt pohybuje. Tato vlastnost způsobuje, že se obrázek zobrazuje v různých webových prohlížečích jinak. Například Internet Explorer neumí zobrazit animace a tedy nepodporuje uvedený element animateMotion. Obdélník bude v tomto případě zobrazen v levém horním rohu. Ostatní prohlížeče, které podporují animace již obrázek zobrazí korektně, tedy bude pozice elementu závislá na rodičovském elementu.

4.2.4 animateColor

Tento element již je zastaralý a W3C jej doporučují nahradit elementem animate. Element animateColor mění barvu požadovaného elementu ve stanoveném čase. Použití tohoto elementu je uvedeno níže, viz. Výpis 8.

```
<circle cx="60" cy="60" r="50">
       <animateColor attributeName="fill"
                     attributeType="XML"
                     from="black"
                     to="red"
                     dur="6s"
                     repeatCount="indefinite"/>
    </circle>
```

Výpis 8: Změna barvy výplně

V příkladu výše se změní výplň kružnice z černé na červenou. Stejnou operaci lze vytvořit pomocí elementu animate.

4.2.5 animateTransform

Element animateTransform umožňuje deformovat objekt v nastaveném čase. SVG nabízí tyto základní transformace:

translate - přesun objektu

rotate - rotace objektu

scale - změna měřítka

matrix - transformace

Typ transformace je povinný a zapisuje se do atributu type. Níže je zobrazen příklad použití tohoto elementu, viz. Výpis 9.

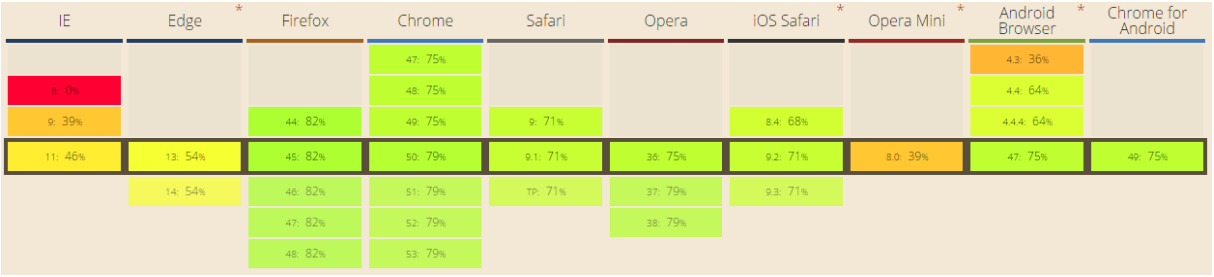
```
<rect x="20" y="20" width="40" height="40"
      style="stroke: #ff00ff; fill: none;" >
    <animateTransform attributeName="transform"
                      type="scale"
                      from="1 1" to="2 3"
                      begin="0s" dur="10s"
                      repeatCount="indefinite"
          />
</rect>
```

Výpis 9: Transformace objektu

Na příkladu výše se v průběhu 10 sekund mění poměr stran obdélníku, resp. z čtverce, který má poměr stran 1:1 (from) na obdélník s poměrem stran 2:3 (to). Typ transformace je definován pomocí atributu type.

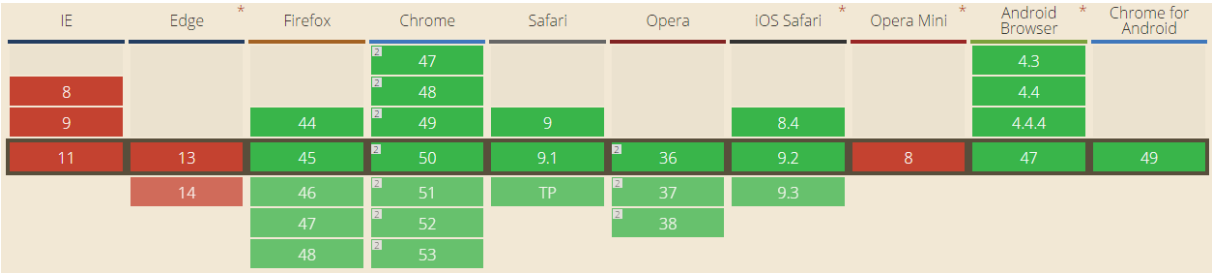
4.3 Podpora formátu SVG

Přes všechny výhody SVG není podpora tohoto formátu příliš rozšířená. Největší podporu zajišťují prohlížeče postavené na jádře Gecko - Mozilla Firefox a Webkit - Opera a Google Chrome. Níže je zobrazena tabulka (viz. Obrázek 8) s procentuální podporou standardu SVG u každého prohlížeče.



Obrázek 8: Úroveň podpory formátu SVG

Po stránce podpory animací je již situace lepší. Animace ve formátu SMIL neumí zobrazit pouze prohlížeč Internet Explorer, Edge a Opera Mini, jinak všechny ostatní běžně webové prohlížeče animované SVG zobrazují bez větších problémů, viz. Obrázek 9.

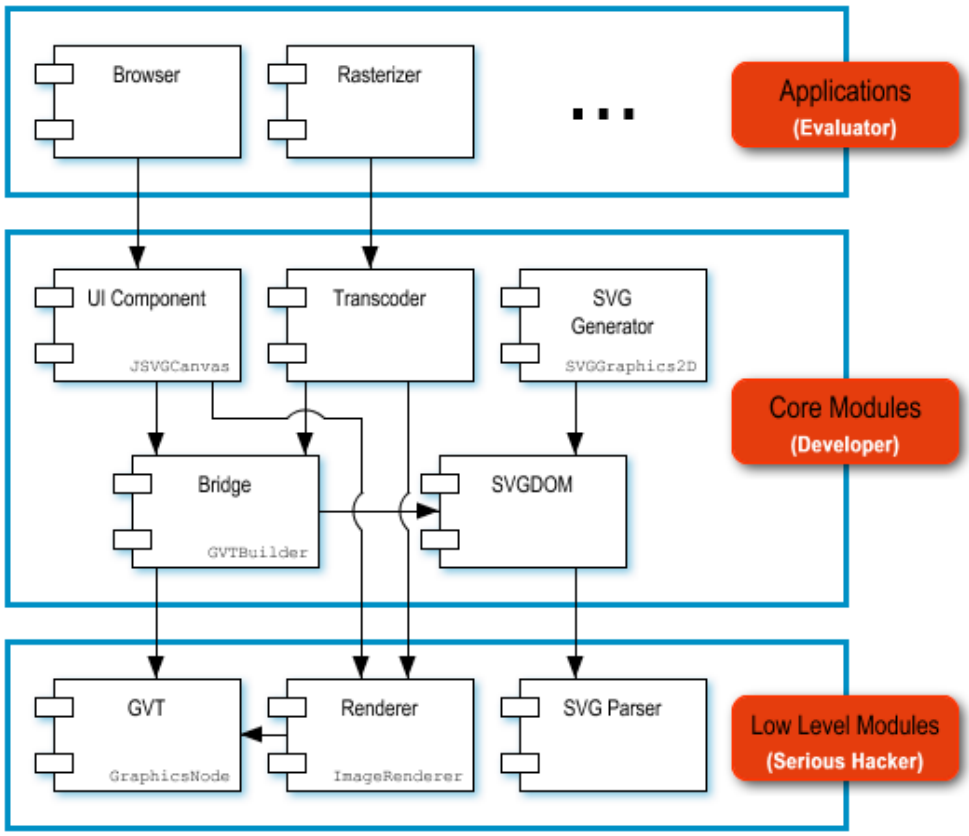


Obrázek 9: Podpora deklarativních animací

Oba diagramy pocházejí ze služby Can I Use[2], která spravuje úroveň podpory různých technologií ve webových prohlížečích. Výsledky jsou průběžně aktualizovány. Výše zobrazené diagramy pocházejí z dubna roku 2016.

5 Apache Batik

Apache Batik je rozsáhlá knihovna pro programovací jazyk Java, která zajišťuje podporu pro čtení, zápis a editaci SVG formátu. Aktuální verze obsahuje téměř kompletní podporu verze 1.1 SE. Batik je poměrně rozsáhlou knihovnou, celková velikost aktuální verze je přes 10 MB. Knihovna je tvořena třemi základními moduly[24][15], viz. Obrázek 10.



Obrázek 10: Moduly Apache Batik

5.1 Moduly knihovny Apache Batik

5.1.1 Applications

Tento balík je rozdělen na části SVG Browser (Sguiggle), SVG Pretty-Printer, SVG Rasterizer a SVG Font Converter. Jedná se o ukázkové aplikace využívající Apache Batik.

Sguiggle

Jednoduchý prohlížeč SVG souborů. Nabízí téměř plnou podporu standardu a je schopen zobrazit i animované elementy. Kromě samotného zobrazení (včetně přiblížení a rotace) umožňuje i konverzi do rasterových formátů jako je JPEG, TIFF a PNG.

SVG Rasterizer

Demonstruje použití komponenty Transcoder z balíku Core Modules. Účelem SVG Rasterizer je konverze SVG souboru do bitmapových formátů JPEG, TIFF, PNG a také PDF. Konverze do PDF formátu vyžaduje Apache FOP knihovnu.

SVG Font Converter

Ne vždy je možné zajistit, aby na cílovém zařízení byl dostupný požadovaný font. Tento problém řeší SVG Font Converter, který font ve formátu TrueType převede do nativního SVG formátu. V tomto případě, se písmo uloží formou definice, tedy za pomocí elementu <defs>.

SVG Pretty Printer

SVG Pretty Printer je modul, který zajišťuje formátování (normalizaci) SVG souboru. Pomocí tohoto modulu můžeme stanovit délku tabulátoru, případně sjednotit řádkování.

5.1.2 Core Modules

Modul Core Modules obsahuje moduly, které tvoří jádro Apache Batik.

SVG Generator

Komponenta, která převádí grafiku na SVG formát. V programovacím jazyce Java je grafika reprezentována třídou Graphics2D. Apache Batik tuto třídu rozšiřuje třídou SVGGraphics2D. Díky této třídě jsme schopni zapsat standardní grafiku (například libovolná implementace rozhraní Shape) do SVG formátu.

SVG DOM

Komponenta, která implementuje SVG DOM API definované SVG specifikací. Umožňuje pracovat s celou strukturou SVG souboru.

JSVGCanvas

Swing komponenta (rozšiřuje třídu JComponent), která umožňuje zobrazit SVG soubor, včetně jednoduchých interakcí jako přiblížení (zoom), rotace, apod.

Transcoder

Komponenta, která zajišťuje konverzi mezi formáty. Současná verze Apache Batik obsahuje implementaci pro konverzi do JPEG, PNG, TIFF a PDF formátů.

Bridge

Zpravidla nebývá užíván přímo, ale prostřednictvím jiné komponenty. Bridge je zodpovědný za vytváření jednotlivých objektů (GVT), které odpovídají SVG elementům.

5.1.3 Low Level Modules

Je tvořeno základními moduly, které využívá nadřazená část „Core modules“.

Graphic Vector Toolkit

Balík tříd, který zajišťuje popis vektorové 2D grafiky za pomocí stromu objektů, které reprezentují grafické elementy.

Renderer

Modul, který zajišťuje renderování SVG dokumentu, převedeného do GVT.

SVG Parser

Modul zajišťující parsování (převod do objektové podoby) SVG souboru.

5.2 Základní práce s Apache Batik

5.2.1 Zobrazení SVG souboru

Apache Batik nabízí dvě základní možnosti renderování načteného souboru. První je použití implementace rozhraní Transcoder. Kromě vlastní implementace lze využít i stávající připravené třídy pro export do formátu JPEG, PNG, TIFF a PDF.

Dále je k dispozici Swing komponenta JSVGCanvas, která je schopna zobrazit SVG soubor včetně animací. Vstupem může být instance třídy SVGDocument případně cesta k souboru, přičemž samotné načtení a parsování zajistí komponenta sama.

5.2.2 Proces načtení a renderování scény

Průběh renderování lze sledovat pomocí základních připravených posluchačů[4][16]. V níže uvedeném seznamu se posluchačem rozumí rozhraní a událostmi jsou myšleny jeho metody. Každé z těchto rozhraní obsahuje i adaptér pro případ, kdy není nezbytné odchytávat všechny události.

SVGDocumentLoaderListener

Tento posluchač poskytuje několik metod pro sledování událostí při načítání SVG souboru. Pokrývají fázi sestavování DOM stromu.

GVTTreeBuilderListener

GVTTreeBuilderListener poskytuje přístup k procesu sestavování stromové struktury všech grafických elementů. Jednotlivé uzly jsou reprezentovány třídou GraphicsNode, které umožňuje přístup k jednotlivým metodám pro získání obrysu, detekce překrývání, apod.

SVGLoadEventDispatcherListener

Posluchač SVGLoadEventDispatcherListener pokrývá události, které nastávají při načítání dynamických SVG souborů, tedy takových, které obsahují animace. K dispozici jsou čtyři události implementující toto rozhraní. Tyto události popisují start a dokončení načtení informací o všech animacích. Popis animací je pak uložen v kolekci objektů typu TimedElement ve třídě SVGAnimationEngine.

TimegraphListener

Výše zmíněnou fázi přidávání animací lze sledovat za pomocí posluchače TimegraphListener. V tomto rozhraní je celkem 11 metod (událostí). Pro běžné účely jsou však důležité pouze dvě a to elementAdded a elementRemoved. Pomocí odchycení události elementAdded lze získat objekt TimedElement, který obsahuje informace o délce a času počátku animace. Další události popisují editaci jednotlivých animací ať už z pohledu jejich časování anebo aktivace, resp. deaktivace.

[4]Posluchačem neboli listenerem je zde myšlena implementace rozhraní, která obsahuje metody, jež informují o události.

U těchto událostí se však nepodařilo nasimulovat jejich vyvolání. Nicméně pro účely této práce nejsou nezbytné.

GVTTreeRendererListener

Po načtení a připravení všech animací nastává renderování obrazu. Průběh renderování lze kontrolovat za pomocí událostí z rozhraní GVTTreeRendererListener. Událostí je k dispozici celkem pět. Popisují přípravnou fázi, začátek renderování a jeho dokončení, případně události pro chybový stav anebo zrušení renderování.

UpdateManagerListener

Tento posluchač nabízí řadu metod pro zachytávání událostí typu UpdateManagerEvent. Tyto události je nezbytné znát pro ovládání běhu animací. V dokumentaci ApacheBatik jsou sice zmíněny, ale jejich popis je velice stručný a bylo nezbytné jejich chování zanalyzovat za pomocí zdrojových kódů knihovny a krokování kódu při zobrazení animovaného SVG. Zdrojem těchto událostí je třída UpdateManager, která zajišťuje samotné renderování animací. UpdateManagerEvent obsahuje informace o právě prováděném kroku animace. Lze tedy získat informaci o tzv. „dirty area“, což znamená oblast, kde došlo ke změně obrazu, výsledný obraz typu BufferedImage a logickou hodnotu udávající, zda došlo k úspěšnému vykreslení.

UpdateManager nabízí tyto události:

managerStarted(UpdateManagerEvent updateManagerEvent)

Událost managerStarted nastává po inicializaci UpdateManager. Nejprve se pro kořenový element RootGraphicsNode nastaví UpdateTracker, který sleduje změny v grafice. Pak nastane událost managerStarted a logická proměnná started je nastavena na true. UpdateTracker se dále využívá v případě vykreslování a souvisí s následujícími událostmi. Například v metodě repaint se kontroluje pomocí UpdateTracker pokud nastala změna (byla provedena animace), v případě, že ne tak není provedeno vykreslení, v opačném případě se vykreslí scéna a nastanou události informující o prokreslení.

managerSuspended(UpdateManagerEvent updateManagerEvent)

ManagerSuspended informuje o pozastavení vykreslování animace. Tato událost nastává ihned po zavolání metody suspend, pakliže vykreslování právě probíhá, v opačném případě není událost vyvolána.

managerResumed(UpdateManagerEvent updateManagerEvent)

Pokud je UpdateManager pozastaven za pomocí metody suspend a je následně obnoven za pomocí metody resume, tak nastane událost ManagerResumed.

managerStopped(UpdateManagerEvent updateManagerEvent)

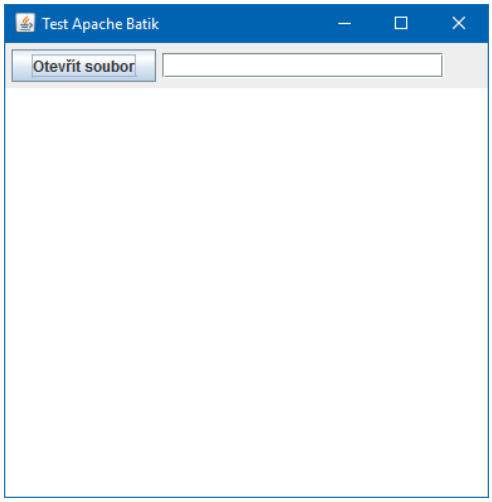
Událost managerStopped je vyvolána po zavolání metody interrupt, která zastaví vykreslování a odstraní veškeré vlákna z fronty pro zpracování. Z tohoto stavu nelze animování obnovit a je nutné provést opětovnou inicializaci třídy UpdateManager.

updateStarted, updateCompleted a updateFailed

V průběhu renderování animace jsou vyvolány dvě události. Jedná se o updateStarted, která informuje o začátku renderování, v případě úspěchu je pak vyvolána další událost a to updateCompleted. Při překreslování může nastat chyba, v tomto případě nastává událost updateFailed.

5.2.3 Využití v praxi

Mějme jednoduchou aplikaci, která bude obsahovat vykreslovací plátno, textové pole a tlačítko pro načtení souboru. Ukázka aplikace, viz. Obrázek 11.



Obrázek 11: Testovací aplikace

Po kliknutí na tlačítko Otevřít soubor se zobrazí standardní dialog pro otevření souboru, využijeme tedy JFileChooser. Po vybrání SVG souboru získáme řetězec obsahující absolutní cestu. Nyní inicializujeme vykreslovací plátno, které je reprezentováno komponentou JSVGCanvas. Jak již bylo zmíněno výše, můžeme využít metodu setURI pro nastavení cesty souboru anebo setSVGDocument pro vložení instance třídy SVGDocument, viz. Výpis 10 níže.

```
String parser = XMLResourceDescriptor.getXMLParserClassName();
SAXSVGDocumentFactory documentFactory = new SAXSVGDocumentFactory(parser);
Document doc = documentFactory.createDocument(svgFilePath);
svgCanvas.setSVGDocument((SVGOMDocument) doc);
```

Výpis 10: Načtení SVG souboru

Proměnná svgCanvas ve výše zobrazeném výpisu je objekt instance JSVGCanvas. Po zavolání tohoto kódu se ve vykreslovacím plátně zobrazí načtený SVG soubor. Před samotným zavoláním metody setSVGDocument můžeme s objektem doc dále pracovat, například zjistit počet elementů a tento údaj zobrazit do textového pole. V případě, že bychom použili metodu setURI, tak musíme využít posluchače svgDocumentLoaderListener a odchytit událost documentLoadingCompleted, která nás informuje o načteném a zparsovaném SVG souboru. Ukázka kódu viz. Výpis 11.

```
svgCanvas.addSVGDocumentLoaderListener(new SVGDocumentLoaderAdapter() {
   @Override
   public void documentLoadingCompleted(SVGDocumentLoaderEvent e) {
      info.setText(
         e.getSVGDocument()
          .getRootElement()
          .getChildNodes()
          .getLength() + " elements");
   }
});
```

Výpis 11: Využití události documentLoadingCompleted

Nyní víme, kdy máme přístupný zparsovaný soubor. Řekněme, že načtený soubor obsahuje komplikovanější obrazce. Komplikovanějším obrazcem se myslí tvary složené z uzavřené cesty. Takové tvary na rozdíl od obdélníku nebo elipsy nemají atributy pro specifikaci šířky a výšky. Chceme-li přidat do textového pole informaci o elementu s největší šířkou, tak budeme muset využít přístup ke stromové struktuře GVT (Graphic Vector Toolkit), která obsahuje elementy implementující rozhraní GraphicsNode. Z těchto elementů lze pak získat objekt implementující rozhraní Shape, který nám umožňuje přistupovat k rozměrům daného tvaru. Pro zjednodušení ukázky získáme pouze první element a vypíšeme jeho šířku.

GraphicsNode objekty se inicializují po zparsování dokumentu, tedy po vyvolání události documentLoadingCompleted. Jak již bylo zmíněno výše, proces sestavování GVT stromu popisuje posluchač GVTTreeBuilderListener, kterého využijeme v následující ukázce, viz. Výpis 12.

```
svgCanvas.addGVTTreeBuilderListener(new GVTTreeBuilderAdapter() {
   @Override
   public void gvtBuildCompleted(GVTTreeBuilderEvent e) {
      RootGraphicsNode rootNode = e.getGVTRoot().getRoot();
      CompositeGraphicsNode canvasNode = (CompositeGraphicsNode) rootNode.get
          (0);
      ShapeNode shape = (ShapeNode) canvasNode.get(0);
      info.setText("width: " + shape.getShape().getBounds2D().getWidth());
   }
});
```

Výpis 12: Využití události gvtBuildCompleted

Všechny výše popsané posluchače lze použít pro získání informací o statickém obrazu. Budeme-li načítat dynamické SVG obsahující animace, tak budeme potřebovat zjistit délku jednotlivých animací. K těmto údajům lze přistupovat získáváním jednotlivých atributů z elementů před-

stavujících animace. Apache Batik bohužel neobsahuje mnoho způsobů jak přistupovat k informacím o animacích. Jedním z nich je využití abstraktní třídy TimedElement. Bohužel k této třídě se nepřistupuje velmi snadno. Jediný nalezený způsob je využití rozhraní SVGContext. Přístup k animacím je možný až po inicializaci třídy AnimationEngine. Informaci, kdy jsou načteny všechny animace, nám poskytuje posluchač SVGLoadEventDispatcherListener. Ukázka níže (Výpis 13) ukazuje jak získat celkový čas první animace vybraného elementu.

```
svgCanvas.addSVGLoadEventDispatcherListener(new SVGLoadEventDispatcherAdapter()
    {
  @Override
  public void svgLoadEventDispatchCompleted(SVGLoadEventDispatcherEvent
      svgLoadEventDispatcherEvent) {
    Node animatedElement = svgCanvas
        .getSVGDocument()
        .getRootElement()
        .getChildNodes()
        .item(3);
    SVGOMAnimationElement animationElement = (SVGOMAnimationElement)
        animatedElement
        .getChildNodes()
        .item(1);
    SVGAnimationElementBridge animationBridge = (SVGAnimationElementBridge)
        animationElement
        .getSVGContext();
    info.setText("duration: " + animationBridge.getTimedElement().
        getSimpleDur());
  }
});
```

Výpis 13: Využití události svgLoadEventDispatchCompleted

Je důležité zmínit, že hodnota získaná metodou getSimpleDur je vždy hodnota v sekundách. Animace v SVG nemusejí vždy začínat ihned po zobrazení obrazu, ale lze jejich spuštění zpozdit. Bohužel přístup k této hodnotě není ve stávající verzi Apache Batik zrovna přívětivý. Možnosti jsou dvojí. Nejpřímočařejší je získat hodnotu atributu begin přímo ze souboru, přičemž je nutné její hodnotu dále zpracovat, protože nemusí být vždy v sekundách. Druhá možnost je využití stávajícího řešení v Apache Batik. Objekt TimedElement, který získáváme ve výše zobrazené ukázce pomocí metody getTimedElement obsahuje metodu, která umožňuje přístup k poli objektů třídy OffsetTimingSpecifier. Tyto instance pak obsahují údaje o startu animace. Bohužel hodnota má přístup typu protected a není ve třídě obsažena veřejná metoda, která by tuto hodnotu vracela. Jediná možnost přístupu je využití metody toString v této třídě, která vrací

textovou reprezentaci dané hodnoty. Naštěstí je text doplněn pouze znaménkem "+"v případě kladné hodnoty, viz. Výpis 14 níže.

```
protected float offset;

public String toString() {
   return (this.offset >= 0.0F?"+":"") + this.offset;
}
```

Výpis 14: Metoda toString třídy OffsetTimingSpecifier

Výše uvedený posluchač SVGLoadEventDispatcherListener nás informuje o načtení všech animací. Nicméně v případě potřeby je ještě možné využít TimegraphListener, který pak obsahuje událost elementAdded, jež je pak vyvolána po každém přidání animace do třídy SVGAnimationEngine. Jakmile je přidána poslední animace, tak je vyvolána událost svgLoadEventDispatchCompleted, viz. Výpis 15. Nicméně je důležité, aby instance třídy AnimationEngine byla inicializována až po sestavení GVT stromu. Je tedy vhodné tohoto posluchače zaregistrovat po vyvolání události gvtBuildCompleted, kdy je sestaven GVT strom a jsou inicializovány třídy UpdateManager, BridgeContext a AnimationEngine.

```
svgCanvas.getUpdateManager().getBridgeContext().getAnimationEngine().
    addTimegraphListener(new TimegraphAdapter() {
   @Override
   public void elementAdded(TimedElement e) {
      info.setText(e.getElement().getTagName() + "was added");
   }
});
```

Výpis 15: Využití události elementAdded

Nyní je vše připraveno pro vykreslení obrazu. Budeme-li chtít informovat uživatele, že byl úspěšně načten a zobrazen SVG soubor, můžeme k tomu využít posluchače GVTTreeRendererListener. Pro informaci o vykreslení obrazu, lze využít události gvtRenderingCompleted, viz. Výpis 16 níže.

```
svgCanvas.addGVTTreeRendererListener(new GVTTreeRendererAdapter() {
   public void gvtRenderingCompleted(GVTTreeRendererEvent e) {
      info.setText("SVG image was rendered");
   }
});
```

Výpis 16: Využití události gvtRenderingCompleted

Téměř každá aplikace zobrazující animace, musí i nějakým způsobem informovat uživatele o aktuálním čase zobrazené (přehrávané) scény. Jak již bylo zmíněno výše, pro aktualizaci načteného dokumentu slouží třída UpdateManager. V případě animací se aktualizací myslí například posun objektu po ose X. Pro každý krok animace se mění pozice animovaného objektu. O těchto krocích, myšleno počátek aktualizace a dokončení, informuje posluchač UpdateManagerListener. Pro účely zobrazení aktuálního času lze využít události updateComplete, která je vyvolána po aktualizaci a přerenderování scény, viz. Výpis 17 níže.

```
svgCanvas.getUpdateManager().addUpdateManagerListener(new UpdateManagerAdapter
    () {
   @Override
   public void updateCompleted(UpdateManagerEvent e) {
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            SVGAnimationEngine eng = svgCanvas
               .getUpdateManager()
               .getBridgeContext()
               .getAnimationEngine();
            info.setText(eng.getCurrentTime() + "s");
         }
      });
   }
});
```

Výpis 17: Využití události updateComplete

Výsledný čas z metody getCurrentTime je uveden v sekundách s přesností na tři desetinná místa. Stejně jako v případě posluchače TimegraphListener je důležité dát pozor na to, aby byla instance třídy UpdateManager inicializována.

Výše uvedená aplikace je přiložena na CD společně s editorem GLIPS Graffiti. Aplikace byla použita pro testování a analýzu knihovny Apache Batik. Přestože je k této knihovně k dispozici dokumentace JavaDoc, tak mnoho metod není příliš detailně zdokumentováno. K animační části Apache Batik neexistuje takřka žádná volně dostupná literatura. Z těchto důvodů bylo nutné zkoumat zdrojové kódy knihovny a postupně krokovat proces renderování a načítání souboru. Z analýzy knihovny vzešly výše popsané poznatky, které jsou dále aplikovány při implementaci požadovaných funkcí v editoru GLIPS Grafiti.

5.3 Projekty využívající Apache Batik

Apache Cocoon

Open source Framework sloužící pro vývoj webových aplikací založených na XML. Apache Batik je zde použit pro rasterizaci obrázků.

Apache FOP

Program pro sazbu XSL-FO dokumentů do PDF, SVG, PostScript, PNG, apod. Stejně jako v předchozím případě, i zde je Apache Batik využit pro rasterizaci.

The JFreeChart Project

Knihovna, která slouží pro vykreslování grafů a diagramů. Vykreslování zajišťuje Java 2D API. Apache Batik je zde použit čistě pro účely exportu do formátu SVG[13].

Sketsa SVG Editor

Sketsa SVG Editor je multiplatformový vektorový editor, který využívá knihovnu Apache Batik. Jeho schopnosti jsou velice podobné editoru GLIPS Graffiti. Editor podporuje rozšiřování funkcionality prostřednictvím doplňků. Jeden z rozpracovaných doplňků rozšiřuje editor o podporu animací. Podrobnější informace o tomto editoru jsou uvedeny v kapitole Alternativy GLIPS Graffiti 2.3 .

6 Implementace

6.1 Vytváření a vkládání vlastních předdefinovaných tvarů

6.1.1 Analýza

GLIPS Graffiti ve výchozím stavu obsahuje pouze dva předdefinované tvary. Jsou jimi čtverec, resp. obdélník a elipsa. Jediným způsobem jak vytvořit vlastní tvar je použití pera pro kreslení Bézierovy křivky. Tuto křivku lze pak uzavřít, což vytvoří tvar. Pomocí pera lze vytvořit v podstatě jakýkoliv tvar, nicméně pro tyto účely není tento nástroj uživatelsky přívětivý. Takový tvar bohužel nelze uložit a znovu použít, uživatel jej musí vytvořit znovu. Pro tyto účely bude vytvořen nový modul, který umožní vytvořený tvar uložit a po novém spuštění programu znovu načíst.

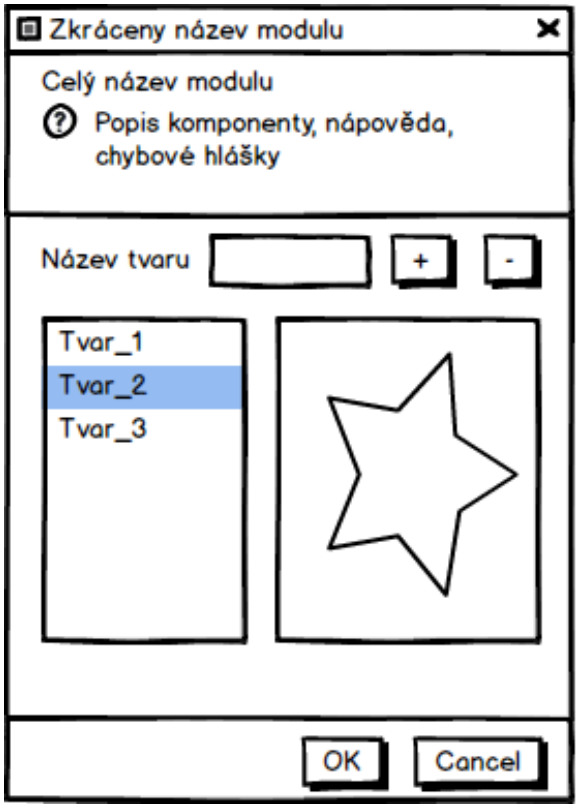
Společně s tímto module bude editor doplněn o další základní předdefinované tvary, typu trojúhelník, pětiúhelník, šipka, apod.

6.1.2 Návrh

Jak bylo uvedeno v kapitole popisující formát SVG, možnosti tvorby nových tvarů jsou v podstatě dvojí. První je vytvořit polyline a druhá je definice cesty, která uzavřením vytvoří tvar. Pro implementaci byla vybrána druhá varianta. Nástroj pro definici cesty je již zabudován, podpora pro uzavírání je také zařízena, zbývá pouze vytvořit třídy, které budou definovat nové tvary. Z důvodu eliminace opakování kódu bude vhodné vytvořit bázovou třídu pro nové tvary, která bude obsahovat základní společné operace předdefinovaných tvarů. Cílem je omezit kód, kterým bude definován nový tvar na minimum a tím umožnit snadné rozšiřování o další nové tvary.

Nyní víme, jak bude reprezentován nový tvar. Dalším krokem bude analýza vyvolání nového tvaru. V příloze je uveden sekvenční diagram popisující výběr nového tvaru a jeho aplikace (Obrázek 27). Stejný postup bude využit v naší implementaci.

Rozhraní pro správu nových tvarů bude obsahovat seznam všech uložených tvarů, po výběru tvaru se zobrazí jeho náhled a bude umožněna aplikace tvaru na plátno. Rozhraní pro přidávání, resp. odebírání nových tvarů bude tvořeno pomocí textového pole pro definici názvu a dvou tlačítek. Na obrázku (Obrázek 12) níže je zobrazen návrh rozhraní modulu.



Obrázek 12: Návrh rozhraní pro správu tvarů

6.1.3 Implementace

V první části implementace začneme nejdříve přidáním nových předdefinovaných tvarů. K definici nových tvarů využijeme křivku neboli cestu. Cesta je reprezentována třídou PathShape, viz. kapitola s popisem modulů. Nejdříve vytvoříme společnou třídu pro všechny předdefinované tvary a to CustomShape. Tato třída bude rozšiřovat třídu PathShape. Umístíme do ní všechny společné metody. Implementace nového tvaru bude umístěna v samostatné třídě. Postup implementace popíšeme na definici tvaru hvězdy.

Podíváme-li se na sekvenční diagram popisující kreslení tvaru (viz. příloha), tak zjistíme, že tvorba tvaru je zajištěna v metodě notifyDrawingAction. Každá třída představující tvar, tuto metodu musí implementovat, protože všechny tyto třídy rozšiřují abstraktní třídu AbstractShape, kde je metoda definována jako abstraktní. Její parametry představují vykreslovací scénu, bod udávající pozici kurzoru, identifikátor modifikátoru události (zda bylo stisknuto levé, pravé tlačítko, či kolečko) a identifikátor události. Identifikátory události jsou definovány v třídě AbstractShape. Identifikátory zajišťují detekcí základních událostí vyvolaných myší. Jedná se o stisknutí tlačítka, tažení kurzoru, dvojklik, apod.

Prvním příkazem v těle metody notifyDrawingAction je získání aktuálního vykreslovacího plátna za pomocí objektu třídy SVGHandle. Následně se již kód větví konstrukcí switch podle proměnné type, která představuje událost myši. Z tohoto důvodu budou dále označeny hodnoty

této proměnné jako události. Obdélník a elipsa zachytávají pouze tři základní události. Komplexnější je pak cesta (PathShape), kde je potřeba odchytávat ještě DRAWING_MOUSE_MOVED, DRAWING_MOUSE_DOUBLE_CLICK a DRAWING_END. V případě vlastních tvarů se budeme inspirovat implementací z třídy RectangleShape. Proces kreslení bude shodný. Po výběru tvaru se klikne kurzorem na plátno, tažením myší (levé tlačítko je stále stisknuto) se bude zobrazovat výsledný tvar a uvolněním tlačítka myši bude kreslení dokončeno. Z tohoto plyne, že konstrukce switch bude shodná, resp. budeme odchytávat shodné události (identifikátory z parametru type). V prvním bloku, kde je odchycena událost DRAWING_MOUSE_PRESSED se zaznamená pozice kurzoru. V kontextu vytváření nového tvaru se jedná o pozici kurzoru, kdy bylo stisknuto tlačítko myši. Další blok DRAWING_MOUSE_RELEASED zajišťuje vytvoření výsledného tvaru, jedná se o poslední krok kreslení. Pro vytvoření tvaru, potřebujeme nejdříve znát jeho rozměr. Toho lze docílit za pomocí počáteční a poslední známé pozice kurzoru. Nejdříve je však nutné ověřit, zda je zapnuto omezení pro čtvercovou selekci a poté zavolat metodu getComputedSquare z třídy EditorToolkit, resp. getComputedRectangle. Výsledná oblast je pak přepočtena do reálných rozměrů (vzhledem k přiblížení plátna) pomocí metody getScaledRectangle() z třídy TransformsManager, jejíž instance je získána z SVGHandle. Následně je již volána metoda createElement s parametry SVGHandle a Rectangle2D, tedy aktuální scéna a rozměry tvaru. Cílem metody je získat aktuálně otevřený dokument, vytvořit element a za využití undo-redo přidat výsledný tvar. Pokračuje se ukončením kreslení, tedy nastavení počátečního bodu na null, odebráním posluchače z plátna a deaktivace kreslícího módu pomocí metody resetDrawing ze třídy AbstractShape. Další z odchytávaných událostí je DRAWING_MOUSE_DRAGGED, která zajišťuje postupné vykreslování tvaru při posunu kurzorem. Samotné vykreslení tvaru je zajištěno ve vnořené třídě GhostShapeCanvasPainter, která rozšiřuje abstraktní třídu CanvasPainter. Tato třída obsahuje dvě členské proměnné shape typu Shape z balíku java.awt a kolekci clips, která je tvořena prvky typu Rectangle2D. Shape představuje tvar, který se má vykreslit, přičemž kolekce udává oblasti, kde dojde k překreslení obrazu. CanvasPainter předepisuje implementaci třídy paintToBeDone s parametrem třídy Graphics2D pro vykreslení tvaru a metodu getClip pro získání kolekce clips. V první části sekce DRAWING_MOUSE_DRAGGED je zkontrolována kolekce clip, zda není null. Pokud není, tak to znamená, že již proběhlo zobrazení tvaru a je tedy nutné z vykreslovacího plátna odstranit předchozí instanci pomocí metody removePaintListener. První parametr vyžaduje objekt typu GhostShapeCanvasPainter a druhý je logická hodnota. V tomto případě je nastavena na false, což zamezí překreslední plátna. Dále je opět zjištěn rozměr tvaru a vytvořen samotný tvar, který je pak nastaven pomocí setteru do objektu GhostShapeCanvasPainter. Následně je přidán posluchač vykreslování za pomocí metody addLayerPaintListener nad plátnem, kde je v prvním parametru specifikována vykreslovací vrstva SVGCanvas.DRAW_LAYER. Dále je vložen samotný objekt pro náhled tvaru a pak logická hodnota nastavena na true, což vynutí překreslení plátna. Celá výše popisovaná metoda může být pro vlastní tvary společná. Jediné co je nutné zajistit, je implementace abstraktní metody, která bude vytvářet tvar dle rozměrů. Tato jediná metoda bude pak obsažena ve třídě

implementující daný tvar. Tímto způsobem výsledný kód maximálně zjednodušíme a umožníme snadné přidávání dalších tvarů. Níže viz. Výpis 18 je zobrazena třída reprezentující tvar hvězdy.

```
public class StarShape extends CustomShape {
    public StarShape(Editor editor) {
        super(editor,"StarShape");
    }

    @Override
    public Shape createShape(Rectangle2D rect) {
        return new Star2D(rect.getX(), rect.getY(), rect.getWidth() / 2 / 3,
            rect.getWidth() / 2, 5);
    }
}
```

Výpis 18: Implementace hvězdy

V případě tvaru hvězda byla využita třída Star2D, která je definována v balíku `org.jdesktop.swingx.geom` a implementuje nezbytné rozhraní Shape. U ostatních tvarů je pak použita třída Polygon2D z balíku `fr.itris.glips.library.geom`. Tato třída rozšiřuje třídu PolyShape2D, která pak implementuje rozhraní Shape. Pomocí třídy Polygon2D lze vytvořit tvar za pomocí bodů, které tento tvar tvoří. Níže je zobrazena ukázka použití této třídy pro implementaci pětiúhelníku (Výpis 19).

```
@Override
    public Shape createShape(Rectangle2D rect) {

        Polygon2D pentagon = new Polygon2D();
        for (int i = 0; i < 5; i++) {
            pentagon.addPoint(new Point2D.Double(
                    (rect.getCenterX() + rect.getWidth() * Math.cos(i * 2 * Math.
                        PI / 5)),
                    (rect.getCenterY() + rect.getWidth() * Math.sin(i * 2 * Math.
                        PI / 5))));
        }
        return pentagon;
    }
}
```

Výpis 19: Implementace pětiúhelníku

Implementaci modulu pro správu nových tvarů začneme vytvořením nové třídy CustomShapeModule, která bude odvozovat třídu ModuleAdapter. Z této třídy pak překryjeme metody

getMenuItems() a getToolItems(). Obě metody vyžadují vytvoření kolekce typu HashMap, kde klíčem je řetězec a hodnotou pro první objekt typu JMenuItem, resp. AbstractButton v případě druhé. Vytvoření těchto objektů přenecháme, stejně jako u jiných již existujících modulů, na třídě ToolsFrame, která je inicializována v konstruktoru. Třída nabízí pouze jeden parametrický konstruktor s parametry Editor editor, String id, String label, JPanel toolPanel. Instanci třídy Editor lze získat z parametrického konstruktoru modulu. Další dva parametry jsou řetězce. První musí obsahovat identifikátor modulu, pomocí něj se získávají položky v menu a popisky, druhý pak představuje popisek v hlavní liště okna modulu. Posledním parametrem je instance třídy JPanel, která představuje představuje kontejner s ovládacími prvky, který bude zobrazen v hlavním okně modulu. Třída ToolsFrame zajišťuje vytvoření plovoucího okna JInternalFrame, který vyplní tímto panelem. Jednotlivé položky v menu a nástrojové liště chceme aktivovat až v okamžiku, kdy bude otevřený soubor a bude možné na plátno vkládat tvary. Toho docílíme získáním tlačítek pomocí metod getMenuItem() a getToolBarButton() z třídy ToolsFrame a zavoláním metody u každého tlačítka setEnabled s parametrem false. Nyní budou tlačítka deaktivována. Aktivací po otevření souboru lze zajistit implementací abstraktní třídy HandlesListener, která obsahuje metody handleChanged a handleContentChanged. Nás zajímá pouze první metoda. Jedná se o událost, která je vyvolána, pokud je otevřen nový soubor, nebo je zavřen, apod. Metoda má v parametrech plátno, kterého se událost týká a kolekci všech pláten v aplikaci. Jednoduchou kontrolou počtu pláten může zajistit aktivaci v případě nenulového počtu, případě deaktivaci pokud je počet roven nula. Tuto implementaci pak vložíme do objektu třídy HandlesManager, ke kterém přistoupíme přes instanci Editor a metodou getHandlesManager(). Nad instancí HandlesManager pak zavoláme metodu addHandlesListener s přidáním našeho posluchače do parametru. Nyní po registraci modulu do souboru modules.xml bude možné okno vyvolat. Nebude však nic obsahovat. Další částí implementace bylo vytvoření rozhraní modulu. K vytvoření rozhraní byly použity standardní komponenty knihovny Swing a jako manažer rozvržení byl použit GridBagLayout. Ze stávajících komponent editoru bylo v plánu použít hlavičku okna. Bohužel tato komponenta využívá dědičnosti z třídy JDialog, přičemž my musíme použít plovoucí okno, tedy JInternalFrame. Z tohoto důvodu byla vytvořena komponenta nová, která je flexibilnější a především je snadno znovupoužitelná. Tato komponenta je definována v třídě DialogHeader. Dědí z třídy JPanel, lze ji tedy jednoduše vložit ať už do dialogového okna anebo do libovolného kontejneru. Umožňuje dynamické vkládání a změnu popisků.

Nyní máme nadefinováno rozhraní modulu. Nejdříve zajistíme ukládání a načítání jednotlivých tvarů. Editor nepoužívá žádnou databázi. Veškeré nastavení, které musí zůstat zachováno se ukládá do registrů systému. Způsob ukládání je popsán v kapitole Správa nastavení aplikace 3.4. Toto úložiště bude výhodným i pro ukládání našich tvarů.

Nejprve naimplementujeme metodu pro naplnění seznamu všech tvarů. Využijeme tedy třídu PreferencesStore, ze které nejdříve získáme seznam všech tvarů. Tohoto docílíme pomocí metody getPreferenceNode, kde do parametru předáváme identifikátor nastavení, což může být v tomto případě název modulu. Metoda vrátí objekt třídy Preferences, ze kterého získáme pole všech

klíčů, pomocí kterých budeme iterovat přes všechny tvary a budeme je přidávat do objektu typu DefaultListModel, který slouží pro přístup k datům, kterými se plní komponenta JList. Výsledná implementace je zobrazena níže, viz. Výpis 20.

```
private void fillShapesList() {
        shapesListModel.clear();
        currentCustomShapes.clear();

        Preferences settings = PreferencesStore.getPreferenceNode(
            preferencesNodeID);
        String[] keys = null;
        try {
            keys = settings.keys();
        } catch (BackingStoreException e1) {
            e1.printStackTrace();
        }
        if (keys != null && keys.length > 0) {
            for (int i = 0; i < keys.length; i++) {
                String customShape = settings.get(keys[i], "");
                shapesListModel.addElement(keys[i]);
                currentCustomShapes.put(keys[i], customShape);
            }
        }
    }
```

Výpis 20: Načtení vlastních tvarů z registrů

Kromě objektu shapesListModel se v metodě ještě používá proměnná currentCustomShapes. Jedná se o kolekci typu Map, která je využívána pro kontrolu duplicit. Výše zobrazená metoda je volána po přidání, odebrání tvaru a při změně viditelnosti okna. Detekci viditelnosti okna nám přistupuje výše zmíněná třída ToolsFrame prostřednictvím metody setVisibilityChangedRunnable, která obsahuje v parametru objekt třídy Runnable, který představuje operaci, která se spustí po změně viditelnosti okna.

Samotné přidávání tvaru je zajištěno v metodě addShapeAction, která je volána ihned po stisknutí tlačítka se znakem plus. Po zavolání metody je nejdříve nutné zkontrolovat, zda je označen element a pak ještě ověřit, jestli se jedná o cestu, kterou lze uzavřít a vložit jako nový tvar. Pro zjištění selekce nejdříve musíme získat aktuální instanci SVGHandle. Kolekci objektů SVGHandle udržuje třída HandlesManager. Její instanci získáme pomocí metody getHandlesManager() z třídy Editor, ke které lze přistupovat pomocí statické metody Editor.getEditor(). Nad objektem HandlesManager pak zavoláme metodu getCurrentHandle(), která vrací objekt třídy SVGHandle, který obsahuje požadovanou metodu getSelection(). Výsledný objekt třídy Selection

zapouzdřuje kolekci vybraných elementů, ke které můžeme přistupovat přes getter getSelectedElements(). Stávající verze bude podporovat pouze ukládání tvarů z jednoho elementu. Budeme tedy brát z kolekce první element. U tohoto elementu nejdříve ověříme, zda se jedná o cestu a pak získáme její definici z atributu "d". Pakliže cesta není uzavřená, uzavřeme ji zadáním znaku "z"na konec získaného řetězce z předchozího kroku. Řetězec pak uložíme do registru s klíčem, který je zadán v textovém poli. Samozřejmostí je validace v každém kroku. Jedná se především o kontrolu selekce, zadaného názvu tvar, apod.

V ukázce lze vidět použití třídy PreferencesStore. Podíváme-li se na její implementaci, tak zjistíme, že jsou implementovány metody pouze pro zapsání a získání hodnoty getPreference a setPreference. Pro potřeby tohoto modulu potřebujeme doimplementovat metodu pro mazání záznamů. Implementace je jednoduchá, viz. Výpis 21 níže.

```
public static void removePreference(String nodeId, String id) {
        try {
            Preferences node = null;
            if (nodeId == null || nodeId.equals("")) {
                node = preferences;
            } else {
                node = preferences.node(nodeId);
            }
            node.remove(id);
        } catch (Exception ex) {
        }
    }
```

Výpis 21: Odstranění nastavení z registru

Metoda v ukázce obsahuje dva parametry typu String, kde první představuje nadřazený uzel a druhým je identifikátor hodnoty. Pakliže první parametr neobsahuje žádnou hodnotu, tak se předpokládá, že hodnota je umístěna v kořenovém uzlu. Tato metoda je pak volána po stisknutí tlačítka se znakem mínus, přičemž nejdříve je získán název vybraného tvaru a pak se ověří, zda název není prázdný. Pokud není prázdný, pokračuje se dále ke smazání hodnoty z registrů a obnovení seznamu tvarů.

V posledním kroku implementace tohoto modulu byl vytvořen náhled vybraného tvaru. K tomuto účelu byla vytvořena samostatná komponenta PreviewElement. Třída této komponenty rozšiřuje třídu JPanel a je tedy umístitelná do libovolného kontejneru knihovny Swing. Vykreslování tvaru pak zajišťuje vnitřní třída Canvas. Třída PreviewElement obsahuje parametrický konstruktor s parametrem logické hodnoty, která udává, zda má být zachován poměr stran elementu anebo se má ignorovat a vyplní celé vykreslovací plátno. Za pomocí tohoto argumentu je inicializována třída Canvas. Pro aktualizaci a nastavení tvaru je přístupná metoda updateShape s argumentem typu Shape.

Vnitřní třída Canvas, která zajišťuje vykreslení tvaru, dědí z třídy JPanel. Vzhledem k potřebě vykreslení se zachováním proporcí elementu anebo jeho ignorováním je k dispozici parametrický konstruktor o dvou argumentech. První je typu Shape a představuje vykreslovaný tvar, druhý je logickou hodnotou, která určuje zachování proporcí. Aktualizaci tvaru zajišťuje metoda se stejným názvem jako ve vnější třídě a to updateShape. Přizpůsobení tvaru, tedy změně velikosti odpovídající náhledu, zajišťuje metoda drawShape s parametrem Shape. V metodě jsou nejprve zjištěny rozměry náhledového plátna a samotného tvaru. Pomocí těchto rozměrů jsou vypočítány koeficienty, pomocí kterých se změní rozměr tvaru s přihlédnutím k nastavení zachování poměru stran a vyplnění plátna. Tvar je transformován pomocí třídy AffineTransform, která představuje API pro afinní transformaci grafiky. Celkem jsou použity tři transformace. Nejdříve je objekt přesunut na okraj plátna, dále se mění jeho rozměr pomocí vypočtených koeficientů a poslední změnou je opět posun na okraj. Tyto transformace se neaplikují postupně, ale jsou sloučeny pomocí metody concatenate a pak aplikovány najednou přes createTransformedShape, kde je výsledkem instance rozhraní Shape. Tento výsledek je návratovou hodnotou metody drawShape. Vykreslení je zajištěno v překryté metodě paintComponent s parametrem Graphics. S pomocí metody drawShape je získán tvar, který bude vykreslen. Pro vykreslení použijeme vyhlazování hran, abychom toho docílili, musíme nejprve parametr přetypovat na Graphics2D, kde je k dispozici metoda setRenderingHint, která umožňuje měnit parametry vykreslování. Jejím zavoláním s parametry RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON) zapneme vyhlazování, které zlepší kvalitu výstupního obrazu. Pomocí metody fill s parametrem typu Shape vykreslíme požadovaný tvar.

6.2 Vkládání animací jednotlivých objektů

6.2.1 Analýza

Základem každého animačního nástroje je přidávání animací k libovolným tvarům. Jak již bylo popsáno v kapitole Animace, SVG formát umožňuje tvorbu animací. Na základě analýzy byla zvolena varianta deklarativního zápisu, tedy SMIL animace.

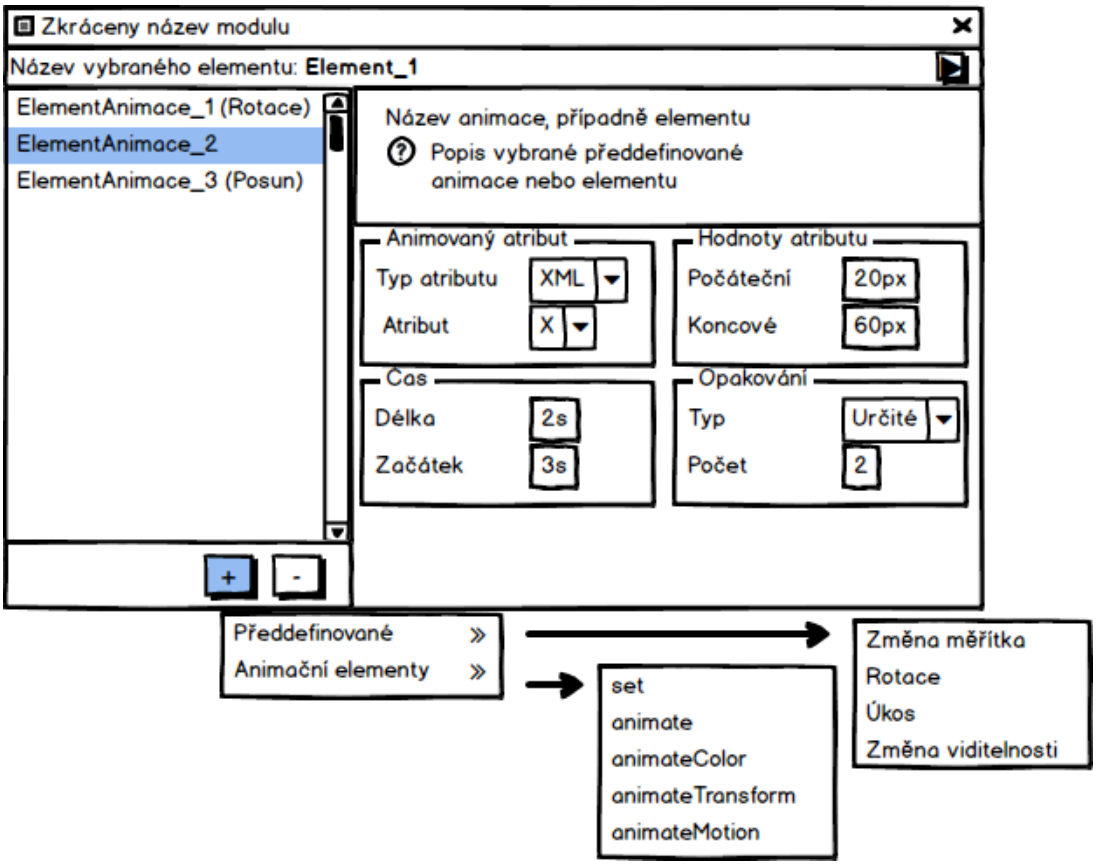
Tvorbu animací bude zajišťovat nový modul. Na výběr budou předpřipravené animace a pak samotné animační elementy. Při implementaci bude kladen důraz na vyšší úroveň abstrakce kódu, aby případné rozšíření o nové předdefinované animace bylo co nejednoduší. Tohoto lze docílit především vyvarování se opakování kódu a používání komponent, jejichž návrh bude dostatečně obecný, aby byly znovupoužitelné[19].

Z předpřipravených animací bude k dispozici změna měřítka, změna viditelnosti, rotace, apod.

Ve stávající verzi editoru není žádná podpora animací SVG. V plnohodnotné verzi lze nalézt okno s časovou osou a plátnem, nicméně to je přizpůsobeno k SCADA/HMI akcím a animace se v něm nepřehrají. Toto okno lze však upravit a podporu animací do něj přidat. Tímto zachováme stávající rozložení a grafiku editoru.

6.2.2 Návrh

Správu animací bude zajišťovat samostatný modul. Tento modul bude v editoru reprezentován dialogovým oknem, které bude zobrazovat všechny animace vybraného elementu. Návrh tohoto rozhraní viz. Obrázek 13 níže.



Obrázek 13: Návrh rozhraní pro správu animací

Dialog je tvořen výpisem všech animací v levé části a jejich detailem v části pravé. Textová reprezentace animací odpovídá samotným elementům, tedy animate, animateTransform, apod. Pokud editor rozpozná předdefinovanou animaci, tak její název uvede do závorky za název elementu. V pravé části okna je zobrazen detail vybrané animace. Detail v horní části obsahuje hlavičku s názvem animace. Pokud nedojde k jejímu rozpoznání, tak bude uveden pouze název elementu, níže je pak zobrazen jeho popis. Pod touto hlavičkou jsou zobrazeny jednotlivé editační prvky pro nastavení vlastností animace. Tyto prvky jsou rozděleny podle jednotlivých kategorií. Přidávání a odebírání animací, pak bude vyřešeno tlačítky pod výpisem animací. Po stisknutí tlačítka pro přidání animace se zobrazí rozbalovací nabídka. V první úrovni je na výběr mezi předdefinovanými animacemi a animačními elementy. Následující úroveň už pak bude obsahovat předdefinované animace, resp. animační elementy. K tomuto rozdělení bylo přistoupeno přede-

vším z důvodu, že pokročilý uživatel tak vytvoří téměř jakoukoliv animaci pomocí animačních elementů a nabídka pouze předdefinovaných by mohla být omezující.

6.2.3 Implementace

Přístup k animacím bude zajišťovat třída Animation. Třída obsahuje členské proměnné, které jsou nezbytné pro práci s animacemi. Jedná se převážně o délku animace, čas začátku, typ atributu, apod. Ke všem těmto proměnným se přistupuje prostřednictvím tzv. getterů a setterů. Díky tomuto způsobu můžeme při každé změně hodnoty vytvořit akci pro vrácení předchozí hodnoty, případně opakování akce. Níže je zobrazena ukázková implementace pro atribut AttributeType, viz. Výpis 22.

```
@Override
    public void updateAttributeType(final AttributeType attributeType) {
        if (attributeType == null) return;
        final AttributeType previousValue = this.attributeType;
        this.attributeType = attributeType;

        if (attributeType != previousValue) {
            final SVGHandle currentHandle = Editor.getEditor().getHandlesManager
                ().getCurrentHandle();

            final Runnable executeRunnable = () -> animationElement.setAttribute
                ("attributeType", attributeType.toString());

            final Runnable undoRunnable = () -> animationElement.setAttribute("
                attributeType", previousValue.toString());

            addUndoRedoAction(executeRunnable, undoRunnable, "AttributeType",
                currentHandle);
        }
```

Výpis 22: Aktualizace atributu

Pro každý z atributů je tato metoda téměř stejná. Samozřejmostí je kontrola, zda argument není null a pak je získána aktuální hodnota atributu. Vzhledem k tomu, že k této hodnotě bude přistupováno z vlákna, tak musí být definována jako konstanta, tedy za použití klíčového slova final. Dále je provedena kontrola, zda se nová hodnota liší od té předešlé. Nemá smysl vytvářet novou undo-redo akci, když se hodnota neliší. Každá akce v undo-redo modulu musí být vložena do nového vlákna (Runnable), takřka u všech atributů se jedná o příkaz na jeden řádek a právě zde je velmi výhodné použití lambda výrazu z Java verze 8. Tímto docílíme větší přehlednosti

kódu a kratší definice metod. Samotná změna hodnoty atributu už probíhá za pomocí metody setAttribute nad objektem SVGOMAnimationElement, který rozšiřuje třídu Element z balíku org.w3c.dom, odkud tato metoda pochází. Poté co jsou vytvořeny objekty pro obě akce (undo a redo) je zavolána metoda addUndoRedoAction, viz. Výpis 23. Tato metoda obsahuje dva parametry typu Runnable. Jedná se o akce pro modul undo-redo. První zajišťuje vykonání akce a druhá tuto akci vrací zpět. Dalším parametrem je řetězec, který popisuje danou akci a pak aktuálně otevřený soubor, tedy instance třídy SVGHandle.

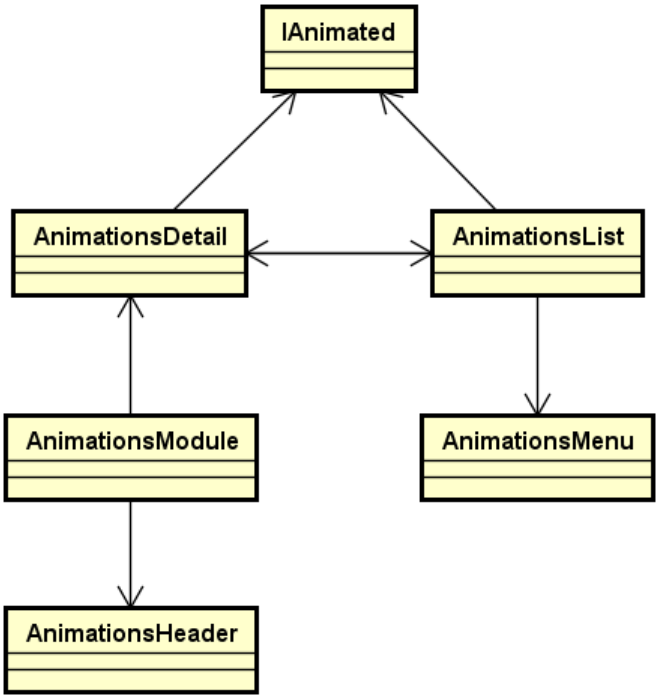
```
private void addUndoRedoAction(Runnable redo, Runnable undo, String caption,
    SVGHandle currentHandle) {
        AnimationController.updateAnimationsList();
        Set<Element> desiredElement = new HashSet<>();
        desiredElement.add(animatedElement);
        desiredElement.add(animationElement);

        UndoRedoActionList actionList = new UndoRedoActionList(caption, true);
        actionList.add(new UndoRedoAction(
                caption, redo,
                undo, redo, desiredElement));
        currentHandle.getUndoRedo().addActionList(actionList, false);

        AnimationController.animationChanged();
    }
```

Výpis 23: Vytvoření undo a redo akce

Tato metoda nejprve volá statickou metodu updateAnimationList z třídy AnimationController, která aktualizuje seznam všech animací. Následně je vytvořena kolekce elementů, jež se účastní akcí z prvního a druhé argumentu metody. Pak je vytvořena instance třídy UndoRedoActionList, která zapouzdřuje akce undo-redo modulu, popisek a elementy, které se této akce účastní. Zavoláním metody addActionList z třídy UndoRedo se zajistí registrace undo-redo akcí. Posledním příkazem metody je zavolání metody animationChanged, která informuje o právě provedené změně v animaci.

Metody updateAnimationList a animationChanged jsou umístěny v třídě AnimationController. Tato třída obsahuje převáženě statické metody, které usnadňují práci s animacemi. Příkladem může být získání seznamu animací.

Rozhraní modulu vychází ze stávajícího modulu Animations and Action v SCADA/HMI části editoru. Při implementaci byl tento modul překopírován a přizpůsoben novému návrhu. Modul pro zobrazení animací je rozdělen do několika dílčích tříd. První z nich, třída Animated, implementující rozhraní IAnimated byla popsána výše. Struktura ostatních tříd a jejich závislosti jsou pak zobrazeny na digramu níže, viz. Obrázek 14.



Obrázek 14: Závislost komponent modulu AnimationsModule

Diagram je velmi zjednodušený, neobsahuje členské proměnné ani metody. Podrobnější verze je přiložena v příloze. Diagram je uveden pro zpřehlednění závislosti jednotlivých tříd, které tvoří modul AnimationsModule. Začněme od nejjednodušší komponenty AnimationsHeader, která zajišťuje zobrazování hlavičky. Implementace této třídy zůstala zachována a na rozdíl od původní třídy HeaderPanel byl odstraněn kód, který zajišťoval podrobnější popisek vybraného elementu. V tomto případě se však jednalo o doplňující SCADA/HMI informace, které nejsou v případě animačního modulu potřebné. Třída obsahuje pouze členské proměnné pro popisky a komponenty JLabel a JButton, které budou umístěny v liště. Vzhledem k jednoduchému rozhraní je použit manažer rozložení BorderLayout. Pro nastavení a získání popisku jsou k dispozici metody getLabel a setCurrentElement. V případě druhé metody se pro zobrazení tučného popisku používají HTML tagy. Třída AnimationsHeader je pak použita pouze v třídě modulu AnimationsModule, kde je volána metoda pro nastavení popisku vybraného elementu. Tlačítko pro zobrazení animací je inicializováno v téže třídě a je předáváno prostřednictvím parametrického konstruktoru do třídy AnimationsHeader.

Další komponentou je AnimationsList. Jak její název napovídá, jedná se o komponentu představující seznam animací. Stejně jako v předchozím případě je využito dědičnosti z třídy JPanel. Původní verze této komponenty AnimationsChooser byla provázána s třídou AnimationObject. Pro reprezentaci animací v této práci je použito rozhraní IAnimated a jeho implementace. Třídu AnimationObject nelze použít a z tohoto důvodu musela být třída AnimationsChooser předělána, aby byla schopna pracovat s rozhraním IAnimated. Úpravy byly provedeny v několika

fázích. První úpravou bylo nahrazení třídy AnimationObject za IAnimated rozhraní. K samotné třídě AnimationObject není přistupováno přímo. V implementaci je využita pouze v kolekci animací a pak prostřednictvím adaptéru. V tomto případě pomocí vnitřní třídy AnimationListItem. Tato vnořená třída obsahuje parametrický konstruktor. Parametrem byla původně třída AnimationObject, ze které se získala textová reprezentace animace. Tento návrh je velice výhodný v případě, že je změněna třída reprezentující animace. Pak stačí pouze upravit vnořenou třídu skrze, kterou se k objektu přistupuje. Samotný proces naplnění komponenty animacemi probíhá pomocí metody setAnimations s parametrem kolekce objektů implementujících IAnimated rozhraní. Uvnitř metody probíhá odstranění aktuálních animací a naplnění novými, které jsou v parametru. Následně se pokračuje voláním metody handleList, která zajistí naplnění členské proměnné listModel typu DefaultListModel instancemi třídy AnimationListItem. Po dokončení operace je vybrána první animace pomocí metody setCurrentAnimation nad objektem třídy AnimationsDetail. Kromě reference na AnimationsDetail je zde i využita třída AnimationsMenu.

AnimationsMenu vychází z původní třídy AnimationsAndActionsMenu. V tomto případě muselo dojít k mnoha změnám. Původní implementace obsahovala funkcionalitu pro vytváření undo-redo akcí, spojených s SCADA/HMI funkcemi. V případě animačního modulu tyto funkce nebudou využity. Třída AnimationsMenu oproti původní implementaci funguje čistě jako nabídka pro výběr nových animací. Rozbalovací nabídka je zajištěna pomocí standardní komponenty Swing JPopupMenu. Základní metodou pro vytvoření nabídky je buildMenuItems s parametrem ActionListener, což je posluchač, který reaguje na výběr položky. Základní úroveň nabídky je tvořena objekty třídy JMenu a každá položka je pak JMenuItem, přičemž popisek je získáván z lokalizačního souboru SVGEditorStrings.properties prostřednictvím metody getLabel, kde je pak využíváno třídy ResourcesManager pro přístup k lokalizaci. Zobrazení nabídky probíhá pomocí metody showPopupMenu, která obsahuje parametry JComponent a Point. První z nich udává komponentu, ve které se má nabídka zobrazit a druhým parametrem je bod, kde bude nabídka zobrazena.

Poslední komponentou, která tvoří animační modul je AnimationsDetail. Tato třída vychází původní implementace v AnimationsPanel, přičemž je zachována dědičnost z třídy JPanel. Jejím úkolem je zapouzdřit seznam animací s tlačítky pro přidání a odebrání animace, tedy zapouzdření třídy AnimationsList a pak zobrazení detailu vybrané animace. Třída je oproti původní implementaci velmi upravena. Obsahuje členskou proměnnou rightPanel třídy JPanel, která představuje kontejner pro zobrazení detailu. Dále je zde již zmíněná instance třídy AnimationsList. Tyto dva prvky jsou rozděleny za pomocí JSplitPane což je Swing komponenta pro zobrazení více prvků oddělených posuvným oddělovačem. Jakmile je dokončeno sestavení rozhraní, tak je prostřednictvím instance AnimationsList získáno menu, kterému je pomocí metody addListner vložen posluchač pro výběr nové animace. Tato metoda všem koncovým uzlům menu vloží pomocí addActionListener daného posluchače, čímž zajistíme odchycení události po kliknutí na položku. Implementace posluchače je reprezentována vnitřní třídou MenuAction v třídě AnimationsDetail. MenuAction implementuje rozhraní ActionListener, které předepisuje

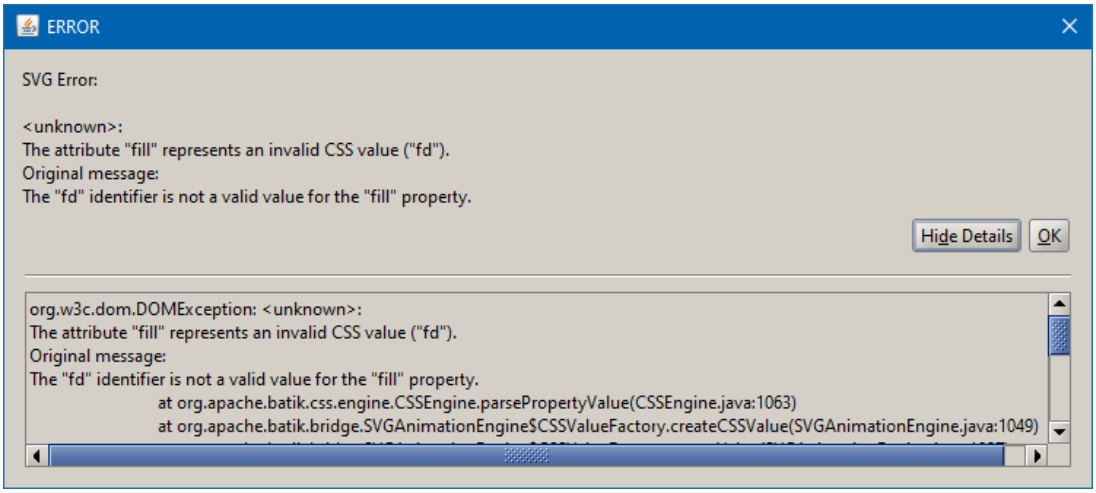
metodu actionPerformed s parametrem ActionEvent. V implementaci této metody je na základě vybrané položky získán detail animace. Detailem se zde myslí rozhraní pro editaci animace, které je obaleno dialogem pro novou animaci, kde jsou pak odchytávány události po kliknutí tlačítka pro potvrzení, případně zrušení akce. Na základě výsledku je vytvořena operace undo-redo s přidáním nového elementu. Logika získávání jednotlivých detailů animací bude popsána v další části. Nyní zbývá ještě doplnit popis metody setCurrentAnimation, která zajišťuje nastavení vybrané animace. Jejím parametrem je objekt typu IAnimated. Po validaci a nastavení hodnoty z parametru do členské proměnné je na základě objektu v parametru získán dialog, tedy detail animace, který je následně zobrazen v pravé části okna.

V dialogovém okně budou zobrazeny dva typy animací. První jsou předdefinované a druhé pokročilé, čímž jsou myšleny samostatné elementy z SVG souboru. Pro každou z těchto animací bude vytvořen dialog. Zde je dialogem myšlena třída odvozující JPanel, kde budou vloženy všechny ovládací prvky animace. Cílem je, aby byla implementace nových předdefinovaných animací co nejjednodušší. Z tohoto důvodu byla vytvořena abstraktní třída AbstractAnimationDialog odvozující JPanel. Tato třída je společná pro všechny dialogy animací. V této třídě jsou obsaženy metody, které jsou společné všem dialogům. Kromě společných metod jsou zde i společné členské proměnné a to instance animace, animovaný element a hlavička s popisem animace. Animace je reprezentována rozhraním IAnimated. Animovaným elementem je instance třídy Element a hlavičkou je již dříve zmiňovaná komponenta DialogHeader, která je použita v rozhraní pro správu vlastních tvarů. V třídě AbstractAnimationDialog jsou pak obsaženy i abstraktní metody, které předepisují jejich implementaci potomkům. Jedná se o metody initialize, compose a setAnimation s parametrem typu IAnimated. Tyto metody slouží pro inicializaci jednotlivých prvků, sestavení rozhraní a nastavení vybrané animace. Každou z těchto operací musí daný dialog vykonávat. Ne každý dialog má však stejné komponenty a implementace tedy bude mírně odlišná, z tohoto důvodu jsou metody definovány jako abstraktní. Třída obsahuje dva parametrické konstruktory. První obsahuje pouze jeden parametr a to instanci rozhraní IAnimated. Tento konstruktor je používán především pro zobrazení aktuální animace. Pro tvorbu nových animací je pak určen druhý konstruktor, který obsahuje navíc parametr selectedElement typu Element. U obou konstruktorů se využívají abstraktní metody, které jsou zmíněny výše. Díky tomuto návrhu výrazně usnadníme implementaci dalších animací, kdy stačí pouze rozšířit tuto abstraktní třídu. Tento krok vynutí implementaci základních metod pro rozhraní a nastavení animace. Konstruktor již pak stačí využít z třídy předka, což je abstraktní třída AbstractAnimationDialog. Níže budou popsány dvě ukázkové implementace. Jedna pro element Animate a druhá bude předdefinovaná animace rotace.

Začněme nejdříve s implementaci dialogu pro element Animate. Dialog pro tento element je v editoru reprezentován třídou AnimateDialog. Rozšířením, tedy použitím klíčového slova extends odvodíme abstraktní třídu AbstractAnimationDialog. Tento krok vynutí implementaci abstraktních metod z předka a to compose, setAnimation a initialize. Zároveň musí odpovídat i konstruktor. Stejně jako ve třídě předka, i zde budou dva konstruktory. Jeden s parametrem

IAnimated, který použijeme pro vytvoření instance dialogu, pokud máme už existující animaci. Druhý bude mít parametr typu Element. Tento konstruktor bude sloužit pro inicializaci dialogu pro vytvoření nové animace. Je důležité zmínit, že předek obsahuje konstruktor s parametry Element a IAnimated. My ale v tomto případě ještě nemáme vytvořenou instanci animace. Ta bude vytvořena uživatelem právě v tomto okně. Z toho důvodu je parametr pouze jeden, instance bude vytvořena v těle tohoto konstruktoru. Jedná se o instanci třídy Animation, což je implementace IAnimated, kde do parametrického konstruktoru vložíme parametr selectedElement a novou instanci SVGOMAnimateElement, která je implementací Element v knihovně Apache Batik. Každá z animací vyžaduje nastavení délky, začátku a opakování. Editaci těchto vlastností zajišťují samostatné komponenty, které jsou univerzální a lze je použít u všech animací.

Pro nastavení časování byla vytvořena třída DurationSettings. Stejně jako u ostatních komponent v této práci, tak i tato třída odvozuje JPanel. Cílem této komponenty je nastavení času startu animace a její délka. Komponenta je tvořena základními Swing komponenty JLabel pro popisek a JSpinner pro nastavení hodnoty. Texty popisků jsou získány za pomocí třídy ResourceManager. Jsou statické a jejich inicializace je umístěna v bloku static, což zajistí, že nastavení hodnoty proběhne pouze jednou a bude společná pro všechny instance třídy. Komponenta obsahuje jeden veřejný konstruktor s parametrem typu IAnimated. Oba ovládací prvky se inicializují za pomocí tohoto objektu. Pouze v případě elementu Set se inicializuje nastavení počátku animace. U obou ovládacích prvků JSpinner je přidán posluchač. Při změně hodnoty je pak zavolána metoda updateDuration resp. updateDurationOffset. Tyto metody pak zajistí aktualizaci hodnoty a vytvoření undo-redo akce. Další komponentou je FromToSettings. Elementy Animate, AnimateTransform a Set obsahují atributy from, to. Tyto atributy mohou mít různé hodnoty. Pro každý z těchto atributů je v komponentě FromToSettings vyhrazeno textové pole pro zadání hodnoty. Implementace je velmi podobná jako v případě DurationSettings. Problém ovšem nastává, kdy uživatel zadá nevalidní hodnotu. V takovém případě nastane výjimka a Apache Batik vyvolá modální okno s chybovou hláškou. Výjimka je vyvolána přímo v kódu knihovny, přičemž ji nelze odchytit v bloku try-catch. Řešením tohoto problému je úprava knihovny a potlačení vyvolání okna s chybovou hláškou. Tento přístup však není příliš vhodný, protože v případě aktualizace knihovny Apache Batik by muselo být pamatováno na tuto úpravu a ta by musela být znovu aplikována. Z tohoto důvodu byl proces validace ze strany knihovny Apache Batik analyzován postupným krokováním a na základě této analýzy byla vytvořena vlastní logika validace, která zamezuje zadání nevalidního vstupu a tedy i vyvolání chybové hlášky.



Obrázek 15: Dialog chyby validace

Výše je zobrazeno modální okno s chybovou hláškou při nevalidním vstupu pro CSS atribut (Obrázek 15). Po kliknutí na tlačítko Show Details se zobrazí zásobník volání, kde zjistíme, že výjimka nastala v třídě CSSEngine a těle metody parsePropertyValue na řádku 1063 při volání metody createValue nad objektem typu ValueManager. Podíváme-li se na implementaci této metody (Výpis 24), tak zjistíme, že je zde odchytávána výjimka a vyvoláváno okno s chybovou hláškou.

```
public Value parsePropertyValue(CSSStylableElement elt,
                                String prop, String value) {
        int idx = getPropertyIndex(prop);
        if (idx == -1) return null;
        ValueManager vm = valueManagers[idx];
        try {
            element = elt;
            LexicalUnit lu;
            lu = parser.parsePropertyValue(value);
            return vm.createValue(lu, this);
        } catch (Exception e) {
//zobrazeni dialogoveho okna s chybovou hlaskou
        } finally {
            element = null;
            cssBaseURI = null;
        }
        return vm.getDefaultValue();
    }
```

Výpis 24: Implementace validace CSS atributu

V implementaci vlastního validátoru použijeme kód, který je ve výpisu výše obalen blokem try. Pro validaci CSS atributu byla vytvořena třída CssAttributeValidator, která obsahuje parametrický konstruktor, jež přijímá animovaný element, atribut a jeho hodnotu. Animovaný element je typu AnimationTarget, což je rozhraní, jehož instanci lze získat z každého elementu animace pomocí metody getParentNode. Název Atributu a jeho hodnota je pak přenášena ve formě řetězce. Validace hodnoty atributu pak probíhá v metodě isValid, která vrací logickou hodnotu. Ve výpisu kódu výše jsou použity metody z třídy CSSEngine, jejich přístupnost je public, nicméně k jejich přístupu potřebujeme instanci třídy CSSEngine. K získání této instance použijeme statickou metodu getCSSEngine z třídy CSSUtilities v balíku org.apache.batik.bridge. Parametrem této metody je objekt typu Element, ze kterého je získán dokument, kterému náleží a z něj je pak získána požadovaná instance CSSEngine.

V případě, že uživatel zadá nevalidní vstup, tak zobrazíme příklad validní hodnoty. Tuto funkcionalitu bohužel také Apache Batik neumožňuje. Nicméně podobným způsobem jako v případě validace, lze zajistit řešení. V metodě, kde je vstup validován se používá rozhraní ValueManager. Podíváme-li se do jeho definice, tak nalezneme metodu getDefaultValue, která vrací objekt typu Value a v této třídě pak máme getStringValue, která vrací řetězec, jež odpovídá výchozí hodnotě atributu.

Nyní víme, jak ověřit validní vstup CSS atributu a zároveň umíme zjistit i jeho validní hodnotu. SVG kromě CSS atributů obsahuje ještě atributy typu XML. U těchto atributů, tak musíme zajistit podobnou funkcionalitu. Pokud vyvoláme chybovou hlášku zadáním nevalidní hodnoty XML atributu v elementu AnimateTransform, tak nastane výjimka v metodě parseValue třídy SVGAnimateTransformElementBridge. V tomto případě šlo o zadání textového řetězce pro animaci rotace. Výjimka nastala při parsování textu na hodnotu float. Kód z metody parseValue nelze použít, je příliš konkrétní a zajišťuje validaci pouze pro hodnoty číselného typu. Posuneme-li se v zásobníku volání o krok zpět do metody createAnimation ve stejné třídě, tak nalezneme již dostatečně obecný kód, který validuje všechny atributy daného elementu, viz. Výpis 25.

```
protected AbstractAnimation createAnimation(AnimationTarget target) {
        short type = parseType();
        AnimatableValue from = null, to = null, by = null;
        if (element.hasAttributeNS(null, SVG_FROM_ATTRIBUTE)) {
            from = parseValue(element.getAttributeNS(null, SVG_FROM_ATTRIBUTE),
                              type, target);
        }
        if (element.hasAttributeNS(null, SVG_TO_ATTRIBUTE)) {
            to = parseValue(element.getAttributeNS(null, SVG_TO_ATTRIBUTE),
                            type, target);
        }
        if (element.hasAttributeNS(null, SVG_BY_ATTRIBUTE)) {
            by = parseValue(element.getAttributeNS(null, SVG_BY_ATTRIBUTE),
```

```
                    type, target);
        }
        return new TransformAnimation(timedElement,
                                      this,
                                      parseCalcMode(),
                                      parseKeyTimes(),
                                      parseKeySplines(),
                                      parseAdditive(),
                                      parseAccumulate(),
                                      parseValues(type, target),
                                      from,
                                      to,
                                      by,
                                      type);
    }
```

Výpis 25: Implementace validace XML atributu

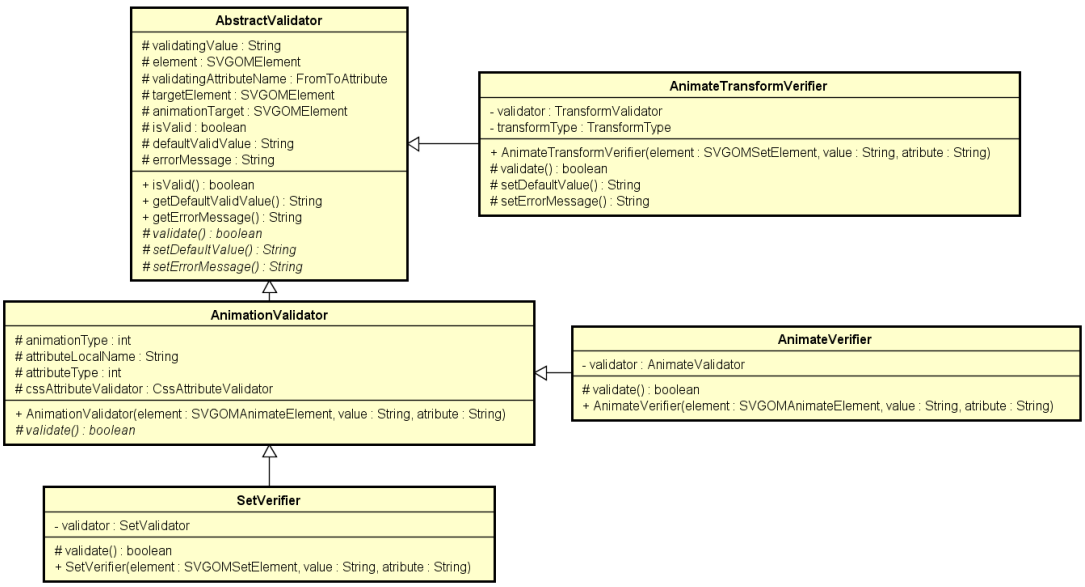
Bohužel tato metoda má přístup protected, takže ji nemůžeme přímo volat i v případě získání instance SVGAnimateTransformElementBridge. Metody parseValue jsou také nastaveny jako protected. Řešením je vytvoření potomka této třídy, kde již tyto metody budou dostupné. Bohužel metody createAnimation a parseValue jsou schopny validovat pouze element AnimateTransform. Celkem tedy využijeme dědičnosti ze tříd SVGAnimateTransformElementBridge, SVGAnimateElementBridge, SVGAnimateMotionElementBridge a SVGSetElementBridge. Výsledná třída validátoru pro element Animate je zobrazena ve výpisu níže, viz. Výpis 26.

```
private class AnimateValidator extends SVGAnimateElementBridge {

        public AnimateValidator(AnimationValidator validator) {
            this.element = validator.element;
            this.targetElement = validator.targetElement;
            this.attributeLocalName = validator.attributeLocalName;
            this.animationTarget = validator.animationTarget;
        }

        private boolean validate() {
            this.eng = (SVGAnimationEngine) Editor.getEditor().
                getCurrentAnimationEngine();
            AnimatableValue animatableVal;
            try {
                if (animationType == 1) {//css
```

```
                return cssAttributeValidator.isValid();
            }
            animatableVal = this.parseAnimatableValue(
                validatingAttributeName.toString());
        } catch (Exception ex) {
            ex.printStackTrace();
            return false;
        }
        return (animatableVal != null);
    }
}
```

Výpis 26: Validace elementu Animate

Každý z těchto validátorů je pak vložen jako vnitřní třída v třídě, která zajišťuje validaci animačního elementu. Tedy pro ověření elementu Set je SetVerifier, pro Animate máme AnimateVerifier apod. Architektura těchto validádatorů je zobrazena na diagramu níže, viz. Obrázek 16.



Obrázek 16: Architektura validačních tříd

Stejně jako v případě CSS atributu i u XML potřebujeme programově zjistit validní hodnotu. Bohužel v tomto případě již není k dispozici žádná třída, která by obsahovala metodu s příkladem validní hodnoty. Nicméně pokud se pokusíme odkrokovat validační proces XML atributu, tak zjistíme, že po zavolání metody createAnimation ze třídy SVGAnimationElementBridge (implementace je pak v třídě odpovídající danému elementu, pro Animate například SVGAnimateElementBridge) je kontrolováno, zda lze atribut animovat. Kontrola probíhá pomocí zjištění

indexu, který označuje typ a poté získáním instance Factory z pole factories v třídě SVGAnimationEngine, viz. Výpis 27.

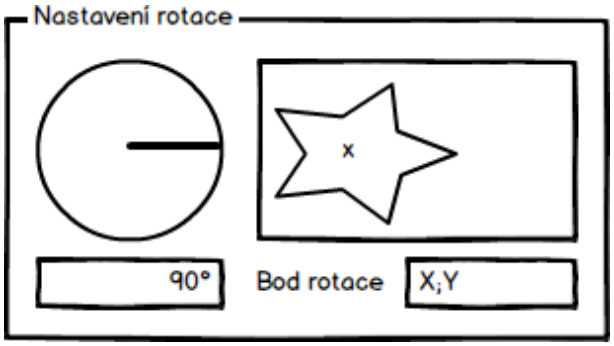
```
public AnimatableValue parseAnimatableValue(Element animElt,
                                            AnimationTarget target,
                                            String ns, String ln,
                                            boolean isCSS,
                                            String s) {
        SVGOMElement elt = (SVGOMElement) target.getElement();
        int type;
        if (isCSS) {
            type = elt.getPropertyType(ln);
        } else {
            type = elt.getAttributeType(ns, ln);
        }
        Factory factory = factories[type];
        if (factory == null) {
            String an = ns == null ? ln : '{' + ns + '}' + ln;
            throw new BridgeException
                    (ctx, animElt, "attribute.not.animatable",
                            new Object[]{target.getElement().getNodeName(), an});
        }
        return factories[type].createValue(target, ns, ln, isCSS, s);
    }
```

Výpis 27: Detekce typu atributu

Každá implementace rozhraní Factory odpovídá typu atributu. Příkladem může být AnimatableLengthValueFactory, AnimatablePointListValueFactory, AnimatableNumberOrPercentageValueFactory, apod. Rozhraní Factory předepisuje metody createValue pro vytvoření instance AnimatableValue. Názvy tříd odpovídají jednotlivým typům, například AnimatableLengthValueFactory se používá pro inicializaci hodnot specifikující délku včetně jednotky, předpokládá se reálné číslo a jedna z podporovaných jednotek (em, ex, px, pt, atd.). Tří SVGAnimationEngine obsahuje v poli factories celkem 51 implementací rozhraní Factory. Postupným testováním byly tyto implementace roztříděny podle typu CSS a XML (viz. výpis v příloze). Na základě specifikace formátu SVG pak byly vytvořeny textové popisky uvádějící validní hodnoty pro typ XML. Pro podmíněný výpis těchto popisků byla vytvořena třída XmlValidValuesHelper, která obsahuje statické metody getDefaultValue a getErrorMessage. Obě metody obsahují parametr typu int, ten představuje index, který odpovídá danému typu. Tento index je možné získat za pomocí metody getAttributeType, viz. výpis kódu výše. Obě metody jsou použity ve třídě AnimationValidator v metodách setDefaultValue a setErrorMessage, kde je nejdříve rozhodnuto,

zda se jedná o CSS anebo XML typ. K tomuto účelu obsahuje Apache Batik metodu hasProperty z třídy SVGOMElement, kde je v parametru očekáván název atributu. Pokud tento název odpovídá typu CSS, tak metoda vrací logickou hodnotu true. V opačném případě se jedná o typ XML a hodnota je false. Podobná třída byla vytvořena i pro element AnimateTransform. Tento element má atribut type, který specifikuje typ transformace. Atributy from a to mohou mít různě specifikované hodnoty a pro nápovědu a chybové hlášky byla vytvořena třída TransformValidValuesHelper, která obsahuje stejné metody jako v předchozím případě. Parametr je ovšem jiný, zde může mít parametr pět různých hodnot a to translate, scale, rotate, skewX a skewY. Tyto hodnoty pak byly vloženy do struktury enum. Na základě těchto hodnot je pak vrácena odpovídající chybová hláška, případně validní hodnota. Tyto metody jsou pak použity ve validátoru elementu AnimateTransform, kde jsou překryty metody setDefaultValue a setErrorMessage, které jsou zmíněny výše.

Výše je popsána validace vstupů a dialogové okno pro animační element. Níže ještě popíšeme jednu z předdefinovaných animací a to rotaci. Pro animaci rotace byla vytvořena třída RotateObject, která rozšiřuje třídu AbstractAnimationDialog. Stejně jako v případě výše popsané třídy AnimateDialog jsou zde členské proměnné typu DurationSettings, RepeatSettings pro nastavení časování a opakování animace. Pro rotaci je použit element AnimateTransform, kde lze pro rotaci definovat počáteční a koncový úhel a bod rotace. Pro snadné nastavení těchto parametrů byla vytvořena komponenta RotateSettings. Návrh této komponenty je zobrazen na obrázku níže (Obrázek 17).



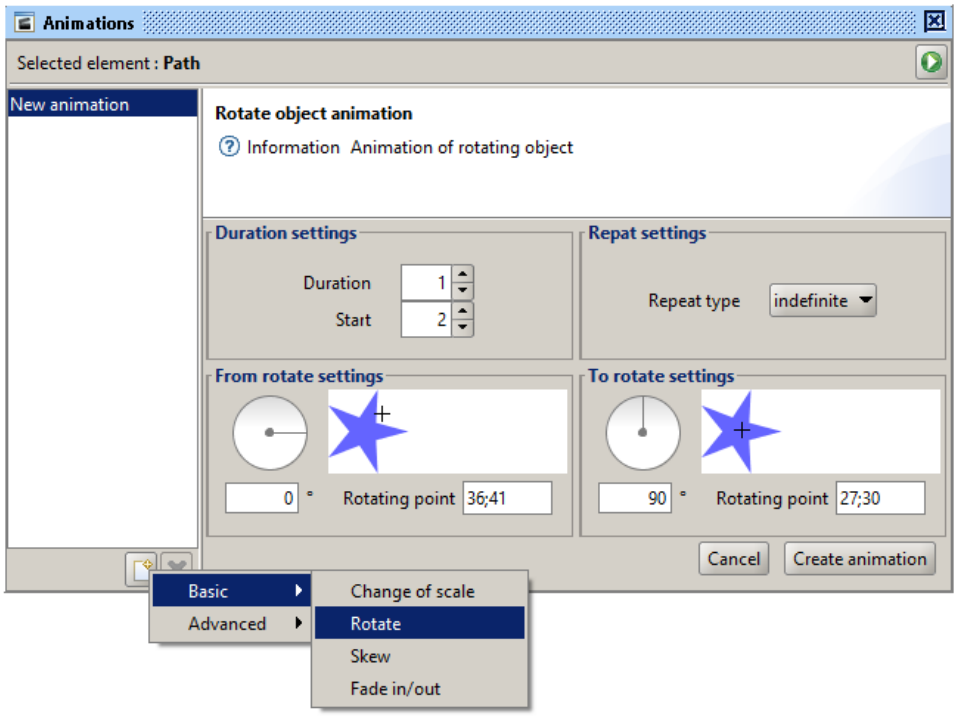
Obrázek 17: Komponenta pro nastavení parametrů rotace

Hlavním prvkem komponenty je otočný ovladač, který umožňuje nastavení úhlu rotace. Kromě tohoto ovladače lze využít i textové pole níže. V pravé části je pak zobrazen náhled animovaného objektu. Náhled je zobrazen se zachováním poměru stran a je v něm formou křížku zobrazen aktuálně nastavený bod rotace. Tento bod lze nastavit ať už kliknutím do náhledu anebo zadáním do textového pole.

Pro náhled elementu máme již vytvořenou komponentu PreviewElement. Tato komponenta byla použita pro správu vlastních tvarů. V této části implementace ji rozšíříme o funkci výběru bodu. Selekci bodu v náhledu zajistíme přidáním posluchače typu MouseListener do konstruk-

toru třídy Canvas. V tomto posluchači budeme odchytávat událost mousePressed. Z parametru metody mousePressed získáme pomocí getPoint instanci třídy Point a tu si uložíme do členské proměnné. Metodou repaint odešleme požadavek na překreslení komponenty. Vykreslení zajišťuje překrytá metoda paintComponent, kde zkontrolujeme, zda je nastaven vybraný bod. Pokud ano, tak jej vykreslíme pomocí dvou čar metodou drawLine.

Bohužel Swing neobsahuje žádnou komponentu, která by byla podobná otočnému ovladači. Z tohoto důvodu byla vytvořena vlastní komponenta, která zajišťuje toto chování. Implementace je umístěna v třídě KnobComponent. Třída odvozuje JPanel, přičemž je překryta metoda paintComponent, kde je vykreslován samotný ovladač. Tvar je vytvořen za pomocí instance Ellipse2D, která je pak vyplněna barevným přechodem GradientPaint. Pohyb ovladače zajišťuje zachycení události mouseDragged pomocí posluchače typu MouseMotionListener. Umístění ukazatele je aktualizováno na základě pozice kurzoru z argumentu metody mouseDragged, který je přepočítán na pozici odpovídající ovladači. Za pomocí metody atan2 z třídy Math je vypočítán úhel theta. Tento úhel je pak dosazen do rovnice kružnice. Tímto výpočtem jsme schopni získat nové souřadnice ukazatele, který je pak překreslen v metodě paintComponent. Výsledná komponenta a zobrazení detailu animace rotace je zobrazena níže na obrázku (Obrázek 18).



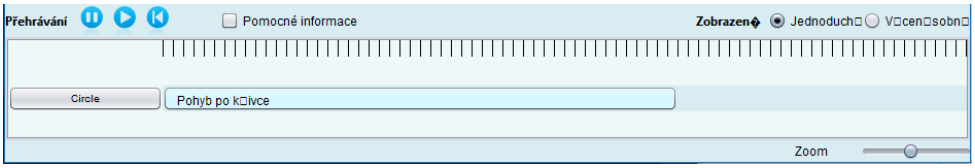
Obrázek 18: Finální modul pro správu animací

Výsledný modul obsahuje kromě výše zmíněných animací ještě animaci změnu měřítka, úkos a změnu viditelnosti. Při jejich implementaci byly využity výše zmíněné komponenty.

6.3 Časová osa

6.3.1 Analýza

Časová osa je velmi komplexní komponenta. Pro Swing existuje mnoho knihoven třetích stran, které představují časovou osu. Na základě doporučení došlo k rozhodnutí použít a rozšířit hotové řešení z bakalářské práce studentů Zdeňka Golda a Martina Golda[8][9]. Jejich řešení je jednoduché a dostatečně flexibilní pro další úpravy.



Obrázek 19: Časová osa před úpravou

Je zřejmé z obrázku (Obrázek 19), že komponenta je velmi jednoduchá. Pro animační účely je ji však nutno upravit. Musí se přizpůsobit pro objekty reprezentující animace v GLIPS Graffiti a pak rozšířit její funkcionalitu. Stávající osa je čistě pro čtení a není možné její prvky nijak upravovat. U všech animací bude možné měnit počátek animace a její délku, pomocí standardní komponenty JSpinner. Daleko pohodlnější bude ovšem přetahovat samotné ukazatele animací na časové ose. Posunem prvku doleva, či doprava změníme čas startu animace. Začátek a konec daného prvku bude možné přetahovat pomocí myši. Tímto bude uživatel schopen zvětšit případně zmenšit šířku daného prvku, což bude reprezentovat délku animace. Změnou šířky u levého okraje se zároveň upraví i čas startu aplikace. Může tedy v jeden okamžik animaci zkrátit a oddálit její start.

Pro tlačítka měnící jednoduché a vícenásobné zobrazení nemáme využití. Budou tedy odstraněny. Na jejich místo se přesune ukazatel přiblížení. Stejně tak nejsou důležité pomocné informace. Ty budou zobrazeny přímo na vykreslovacím plátně aktuálně otevřeného souboru.

Prvky pro ovládání přehrávání budou zobrazeny v jiné grafice, korespondující se vzhledem editoru. Zároveň budou přidána další tlačítka pro ovládání.

Společně s těmito změnami bude upraven i vzhled osy ukazující aktuální čas. Výše uvedená časová osa nemá zobrazeny jednotky, což není moc nepřehledné. Z tohoto důvodu bude časová osa obsahovat celá čísla udávající čas v sekundách. Společně se změnou přiblížení se bude osa zmenšovat a jednotlivé ukazatele budou blíže u sebe. Jakmile nebude dostatek místa, tak se sníží jejich četnost.

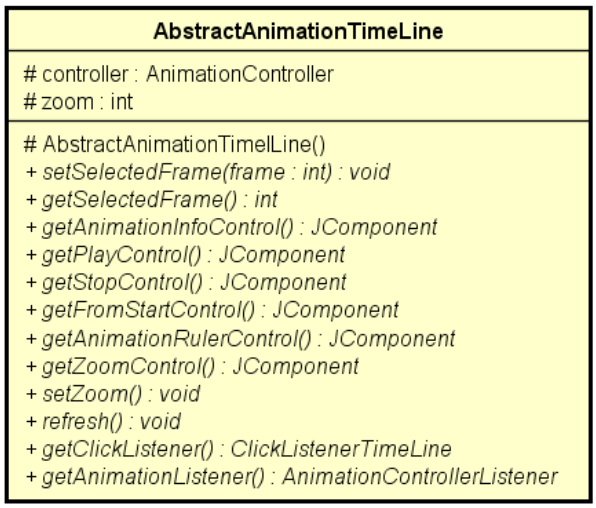
Výsledná časová osa bude umístěna v samostatném modulu, který bude společný pro všechny soubory. Označením aktivního souboru se časová osa aktualizuje a budou zobrazeny animace odpovídající vybranému souboru. Smyslem tohoto modulu je přehlednější zobrazení animací z daného souboru. V tomto zobrazení nebudou viditelné prvky pro ovládání přehrávání. Kromě

tohoto modulu, bude osa také zobrazena ve speciálním okně pro přehrávání animací, kde již boudu vykresleny veškeré prvky pro manipulaci s přehráváním.

6.3.2 Návrh

Stávající architektura komponenty se nebude nijak výrazně měnit. Důležité je pouze přidat podporu pro práci se stávajícími objekty, které obalují animace v GLIPS Graffiti. Z těchto objektů je potřebné získat údaje o časování a název animace.

Všechny grafické části osy využívají dědičnosti ze třídy JPanel. Bázovou třídou je abstraktní třída AbstractAnimationTimeline (viz. Obrázek 20), která je využívána jako výchozí bod pro celou funkcionalitu komponenty. Obsahuje pouze abstraktní metody pro získání jednotlivých komponent časové osy (prvky pro ovládání přehrávání, přiblížení, pravítko, apod.). Implementace třídy a samotné vykreslení časové osy je zajištěno v AnimationTimeLine. Stávající metody v bázové abstraktní třídě nejsou příliš vhodné pro implementaci v GLIPS Graffiti. Cílem je vytvořit modul s časovou osou bez ovladačů přehrávání a pak tuto komponentu vložit do stávajícího okna, včetně tlačítek pro přehrávání. Stávající metody pro získání tlačítek odstraníme a jejich implementaci zajistíme až v konkrétním okně. Toto zajistí vyšší univerzálnost komponenty.

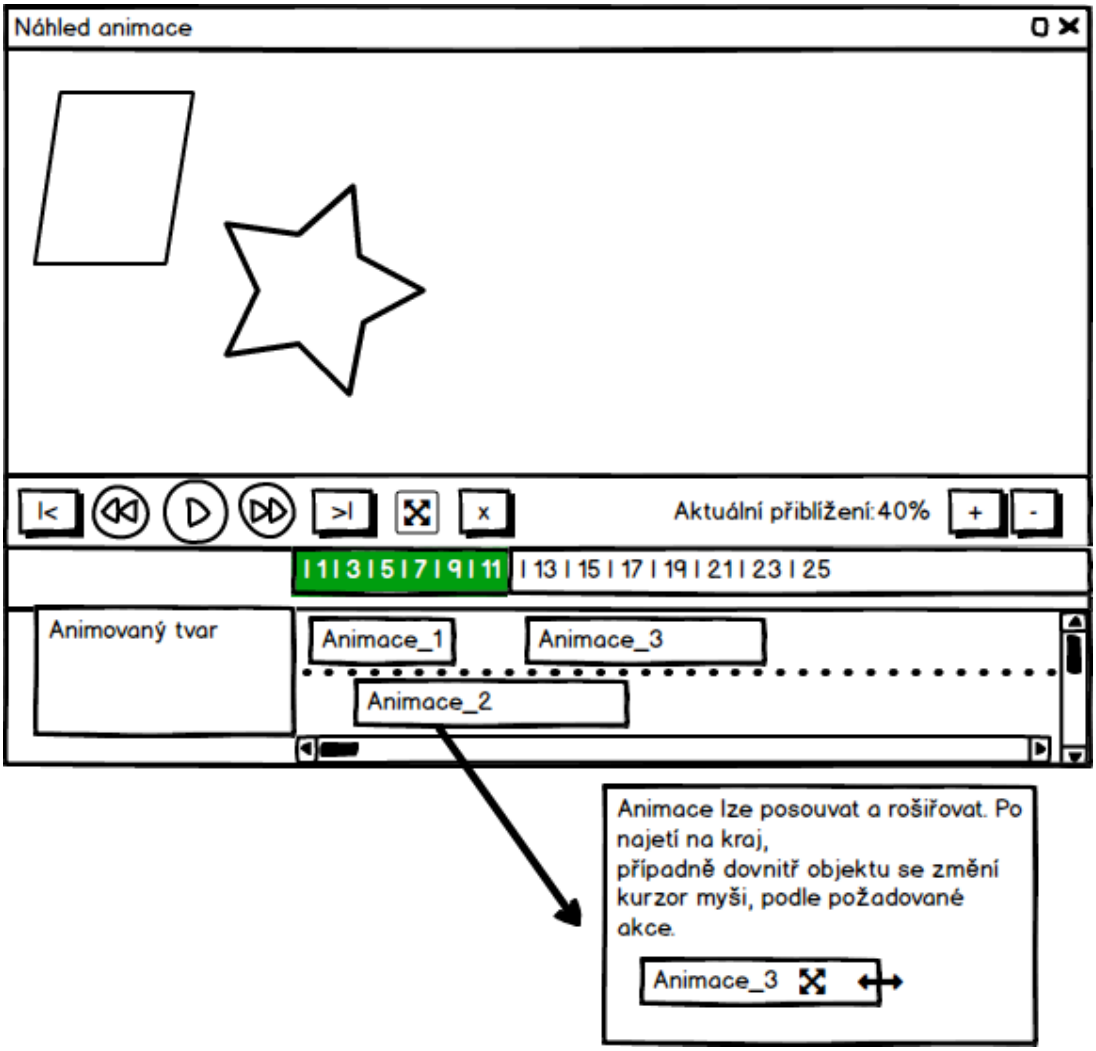


Obrázek 20: Třída AbstractAnimationTimeline

Animované objekty a animace jsou vyjádřeny třídou AnimatedShape, resp. AnimationItem. Obě tyto třídy jsou potomky JButton. Každá z těchto tříd má parametrický konstruktor, ze kterého je získán popisek, jež je nastaven elementu.

Linka, na které jsou umístěny animace je reprezentována třídou AnimationLine. Ukazatele jednotlivých animací jsou zobrazovány pomocí komponenty JButton, která představuje standardní tlačítko. Tuto komponentu rozšíříme o funkčnost pohybu po horizontální ose a změnu rozměru tažením myši.

Pro ovládání přehrávání animace bude použita třída AnimationEngine, která umožňuje nastavení času, zastavení a spuštění animací. Návrh časové osy včetně tlačítek pro ovládání přehrávání je zobrazen níže, viz. Obrázek 21.



Obrázek 21: Návrh náhledu animace

6.3.3 Implementace

Nejdříve byly z abstraktní třídy AbstractAnimationTimeLine odstraněny původní metody pro získání instancí prvků PlayControl, StopControl, FromStartControl, AnimationRulerControl, ZoomControl, dále metody pro nastavení a získání vybraného snímku setSelectedFrame a getSelectedFrame. Poslední z nepotřebných metod jsou getClickListener a getAnimationListener. Důvod odstranění těchto metod je zmíněn výše, ve zkratce se však jedná o rozhodnutí, které zvýší univerzálnost komponenty. Z pohledu architektury je stávající řešení velmi dobře vyřešeno, nicméně chceme-li mít jako komponentu pouze časovou osu, tak musíme odstranit jednotlivé tlačítka a veškeré metody, které k nim náleží. Potomek této třídy je upraven podobným způsobem.

Veškeré proměnné související s ovládacími tlačítky jsou odstraněny. Další změnou je vykreslení dynamického pravítka přímo v této třídě. Dynamičnost tkví v přizpůsobení měřítka na základě přiblížení a nastavení velikosti na základě nadřazené komponenty.

V průběhu přizpůsobování komponenty časové osy byla nalezena třída, která není nikde použita. Jedná se o TimeLineRuler, ta patrně představuje původní měřítko osy. Třída byla smazána, protože její implementace bude zásadně odlišná od té, která je definována v této třídě.

Nyní máme hotový základ časové osy. Dalším krokem implementace je zajištění vykreslení jednotlivých animací a animovaného elementu. Začněme nejdříve elementem. Animovaný element je představován třídou AnimatedShape.

Jak již bylo zmíněno výše, animovaný element rozšiřuje třídu JButton. Jedná se o velmi elegantní způsob, jak jednoduše a přehledně zobrazit element na ose. Tato konstrukce z této třídy dělá zobrazitelnou komponentu ve výsledném panelu. Není nutné implementovat ani překrývat další metody.

Stávající implementace musí být mírně pozměněna. Původní třída používá objekt typu IShape, ze kterého je získána textová reprezentace objektu. My však v editoru pracujeme s XML reprezentací, použijeme tedy třídu Element. Tato změna si vyžádá pouhou změnu typu daného atributu. Jako textová reprezentace slouží atribut ID. Tento atribut je však volitelný a tudíž bylo nezbytné zajistit popisek i pro tento případ. Výsledným řešením je, že v případě chybějícího ID se použije název XML uzlu. Oproti původnímu řešení byl přidán popisek po najetí myší, pomocí metody setToolTipText, který zobrazuje náhled elementu.

Po třídě AnimatedShape upravíme AnimatedItem, která již představuje samotnou animaci. V této třídě bylo nutné provést více změn oproti té předcházející. Nejdříve byla poupravena členská proměnná animation. V aplikaci používáme rozhraní IAnimated, které zajišťuje veškeré informace o animaci. Původní implementace byla odvozena od třídy JButton. V případě, že chceme umožnit posouvání prvků po ose, musíme mezi AnimatedItem a JButton vytvořit další třídu, která bude zajišťovat toto chování. Třída byla pojmenována jako MovableResizableButton.

Třída MovableResizableButton byla implementována s největším důrazem na znovupoužitelnost. Toho lze docílit, pokud se vyvarujeme použitím těsných vazeb.[5] Komunikace s okolními komponenty je umožněna prostřednictvím událostí. Třída nabízí dvě základní události. První je IShapeChangingEvent pro upozornění o probíhající změně tvaru a druhou IShapeChangedEvent, která informuje o dokončené změně. Změnou tvaru se zde rozumí přesunutí po ose anebo změna velikosti elementu.

Veškeré komponenty v Java Swing, které jsou potomky abstraktní třídy Component z balíku java.awt umožňují napojení na události myši. Jsou jimi MouseMotionListener a MouseListener. V případě naší komponenty není nezbytné implementovat všechny události, ale pouze část. Využijeme tedy odpovídající adaptéry MouseMotionAdapter, resp. MouseAdapter. Dané metody v tomto kontextu události implementujeme prostřednictvím překrytí naší implementací. U této komponenty zachytáváme pohyb myší pomocí události mouseMoved, přičemž pokud je

[5] Těsnou vazbou se rozumí závislost na jiné třídě s nízkou úrovní abstrakce.[23][17]

umístěna myš na okraji s tolerancí 5px je změněn typ kursoru na oboustrannou šipku (RESIZE_CURSOR), která upozorňuje uživatele na možnost změny rozměru. Pokud je myš uvnitř komponenty, tak se opět změní kurzor, ale tentokrát na ukazující variantu (MOVE_CURSOR). Kromě změny kurzoru je zde i měněna hodnota logické proměnné, která udává, zda je myš umístěna u levého anebo pravého okraje. Pro tažení objektu je použita událost mouseDragged, která je stejně jako v předchozím případě obsažena v adaptéru MouseMotionAdapter. Jakmile nastane tato událost, tak se za pomocí pozice myši a logické proměnné, která byla zaznamenána v předchozí události vypočítá nový rozměr komponenty, resp. její nová pozice a následně je vyvolána událost informující o měnící se velikosti komponenty. Pro tyto výpočty je nezbytné znát pozici kurzoru, při které došlo k počátečnímu stisknutí levého tlačítka myši. Pozici lze zjistit za pomocí události mousePressed z adaptéru MouseAdapter, která nastává před mouseDragged. Poslední z implementovaných událostí je mouseReleased, kde jsou logické proměnné nastaveny na výchozí hodnoty a následně je vyvolána událost o dokončené změně velikosti.

Nyní se vrátíme zpět k AnimatedItem. Vzhledem k tomu, že tato třída obsahuje členskou proměnnou představující animace, tak právě zde musíme odchytit událost, která nastává po změně prvku v časové ose. Jakmile taková událost nastane, tak si vypočítáme na základě rozměrů prvku, aktuálního přiblížení a konstanty výchozí šířky snímku, délku a počátek animace. Po získání těchto údajů animaci aktualizujeme.

Zobrazení animovaného elementu a jeho animací je zajištěno ve třídě AnimationLine. Tato třída představuje jeden řádek v časové ose. Výšku řádku určuje počet souběžných animací. V této třídě byl změněn typ proměnně, představující animaci. Nachází se zde jedna metoda add(IAnimated animation), která zajišťuje přidání animace k animovánému objektu.

Samotné přiřazení animací k vybranému tvaru je zajištěno v třídě AnimationTimeLineComponent. Tato třída rozšiřuje JPanel. Jedná se tedy o zobrazitelnou komponentu a představuje osu s animovanými objekty a animacemi. Implementaci třídy nebylo nutné stejně jako v předchozích případech příliš měnit. Velká část implementace je umístěna v metodě repaint, která vykresluje komponentu. Nejdříve jsou získány všechny animované objekty, včetně jejich animací a pak jsou zobrazeny. Získávání aktuálních animací je vyřešeno pomocí statické metody getAnimations z třídy AnimationController. Podrobnější popis této třídy a obecně získávání seznamu animací je v kapitole Vkládání animací jednotlivých objektů 6.2. Níže je zobrazen úryvek kódu (Výpis 28), kde se získává seznam animovaných objektů.

```
SVGHandle currentHandle = Editor.getEditor().getHandlesManager().
    getCurrentHandle();
if (currentHandle == null) return;

BridgeContext ctx = null;
SVGOMDocument currentDoc = null;
HashMap<Element, List<IAnimated>> animationCrateHashMap = new HashMap<>();
if (currentHandle != null) {
```

```
    //otevreny svg soubor
    currentDoc = (SVGOMDocument) currentHandle.getCanvas().getDocument();
    ctx = currentHandle.getCanvas().getBridgeContext();
    //seznam animovanych objektu
    animationCrateHashMap = AnimationController.getAnimations(currentDoc, ctx);
}

//iterace po animovanych objektech
for (Map.Entry<Element, List<IAnimated>> entry : animationCrateHashMap.entrySet
    ()) {
    AnimatedShape shape = new AnimatedShape(entry.getKey(), entry.getValue().
        size());
    //umisteni animovaneho tvaru
    //pridani vsech animaci daneho tvaru
    AnimationLine panel = new AnimationLine(s, this);
    entry.getValue().stream().forEach(aR -> panel.add(aR));
    //umisteni animaci
}
```

Výpis 28: Překreslení časové osy

Poslední z komponent časové osy, je samotný ukazatel času, který bude zobrazovat aktuální pozici přehrávání a umožní přetáčení spuštěné scény. Implementace ukazatele je zajištěna v třídě AnimationRuler.

Stávající implementace ukazatele byla zcela změněna. Původní verze byla pouze statická. Na ose chceme zobrazovat časové jednotky na základě přiblížení, aktuální čas a především umožnit uživateli přetáčet animaci kliknutím na požadovaný čas. Z tohoto důvodu jsou v konstruktoru zaregistrovány události updateCompleted, kde se zjistí aktuální čas animace a nastaví se členská proměnná, která se následně použije při překreslení. Přehrávání je pak zajištěno událostí mousePressed, kde získáme pozici kurzoru vůči ose a nastavíme čas přehrávané animaci. Výsledná implementace je zobrazena níže v ukázce kódu, viz. Výpis 29.

```
AnimationsPanel.animationController.addUpdateManagerListener(new
    UpdateManagerAdapter() {
    @Override
    public void updateCompleted(UpdateManagerEvent updateManagerEvent) {
        dur = (float) AnimationController.getTotalTime((SVGOMDocument) Editor.
            getEditor().getHandlesManager().getCurrentHandle().getCanvas().
            getDocument(),
                    Editor.getEditor().getHandlesManager().getCurrentHandle()
                        .getCanvas().getBridgeContext());
```

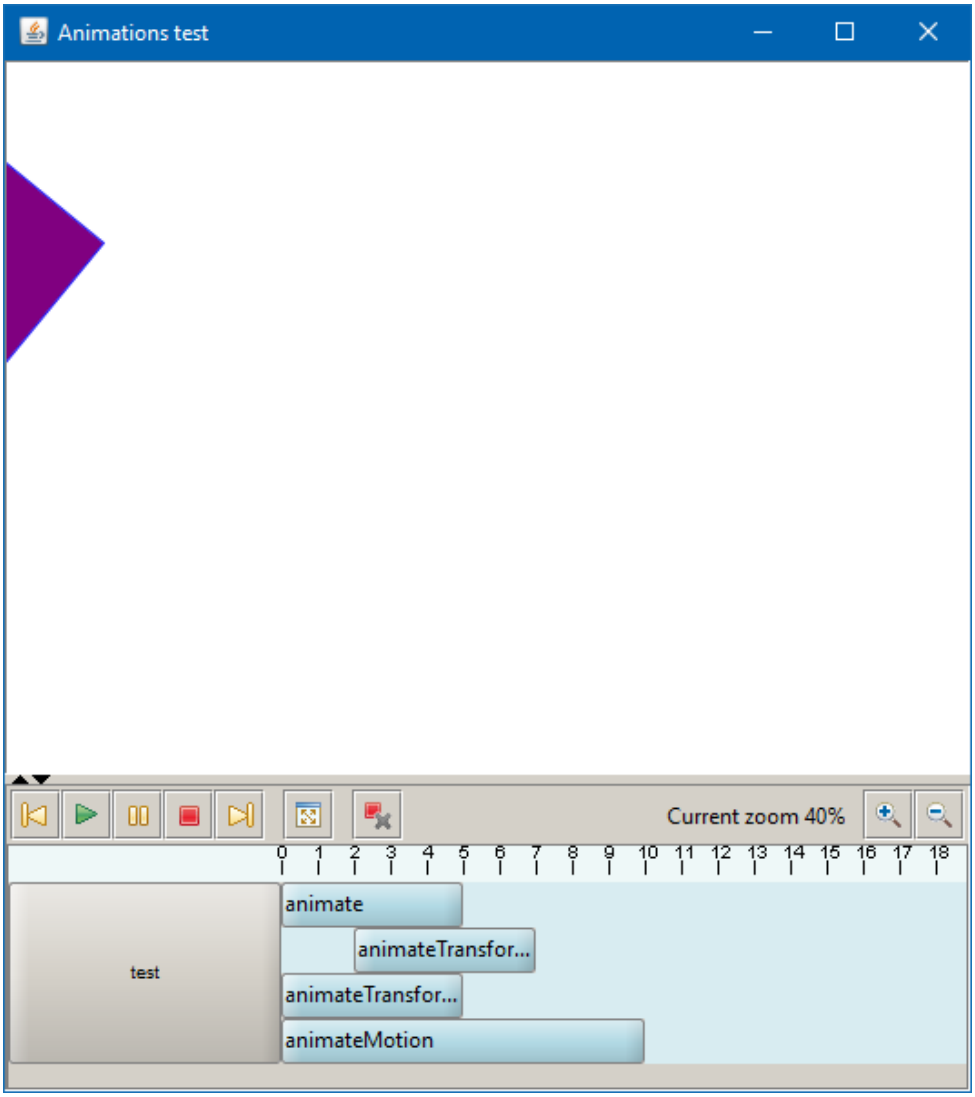
```
      float currentTime = AnimationsPanel.animationController.getCurrentTime();
      if (currentTime <= dur) {
         setSelectedFrame(currentTime);
      } else {
         setSelectedFrame(currentTime % dur);
      }
   }
});

this.addMouseListener(new MouseAdapter() {
   @Override
   public void mousePressed(MouseEvent e) {
      Point2D pt = e.getPoint();
      final double x = pt.getX();
      final float clickedTime = (float) (((x - AnimationTimeLine.
          animationShapeWidth) / (frame_size)));
      AnimationsPanel.animationController.setCurrentTime(clickedTime);
   }
});
```

Výpis 29: Přetáčení animací v časové ose

Komponenta se překresluje v překryté metodě paintComponent(Graphics g) třídy JComponent. Pomocí hodnoty aktuálního přiblížení se vykreslují časové jednotky. Například pro přiblížení nad 30% je zobrazen popisek pro každou sekundu, pro rozsah mezi 20% a 30% se již zobrazuje každá druhá, až po nejnižší hodnotu, kdy se nezobrazují žádné číslice.

Poslední částí je samotné přizpůsobení třídy AnimationTimeLine. Zde je pouze nutné přizpůsobit rozložení pozměněným komponentám.

Plnohodnotná verze časové osy je použita v třídě AnimationsControlPanel, kde je definován panel ovládající animace. Obsahuje tlačítka pro start přehrávání, posun vpřed a vzad, zastavení, pauza a přiblížení. Tato třída byla použita již v původní verzi GLIPS Graffiti. Jejím cílem bylo zobrazení také ovládacího panelu, ale šlo o animace, které byly ovládány SCADA systémem. Finální verze časové osy, včetně ovládání přehrávání je zobrazena níže, viz. Obrázek 22.



Obrázek 22: Finální verze časové osy

6.4 Export animace do video streamu

6.4.1 Analýza

Současné možnosti exportu zobrazené scény v editoru jsou JPG, PNG, BMP a PDF. Všechny třídy zajišťující export do vybraného formátu jsou potomky abstraktní třídy Export, která pomocí abstraktních metod definuje společnou strukturu exportních tříd. Bohužel tato abstraktní třída počítá pouze se statickými obrázky. Například metoda createImage je schopna zapsat pouze instanci třídy BufferedImage. Zde nemůžeme použít metody předka, což znamená, že musíme vytvořit i vlastní metodu pro zápis do souboru.

Současná verze Apache Batik umožňuje dodatečnou implementací exportu do dalších formátů. K tomuto účelu je k dispozici rozhraní Transcoder.

Základní metodou je Transcode, která obsahuje parametry TranscoderInput a TranscoderOutput. Jak již název napovídá, jedná se o objekty, které zapouzdřují vstup exportu a jeho výstup.

Cílem TranscoderInput je získat SVG dokument, který bude exportován. K tomuto účelu slouží celkem 6 parametrických konstruktorů. U GLIPS Graffiti bude využit konstruktor s parametrem Document, nicméně lze použít i InputStream pro přímý vstup ze souboru, Reader, String anebo XMLReader. TranscoderOutput nabízí také několik druhů parametrických konstruktorů. V případě exportu do video streamu použijeme variantu s třídou OutputStream, pomocí které zapíšeme výsledek do souboru.

Další metody z rozhraní Transcoder slouží pro správu nastavení exportu. V tomto případě je nastavení zapouzdřeno v třídě TranscodingHints. Pro přidání nastavení exportu slouží metoda

void addTranscodingHint(TranscodingHints.Key key, Object o)

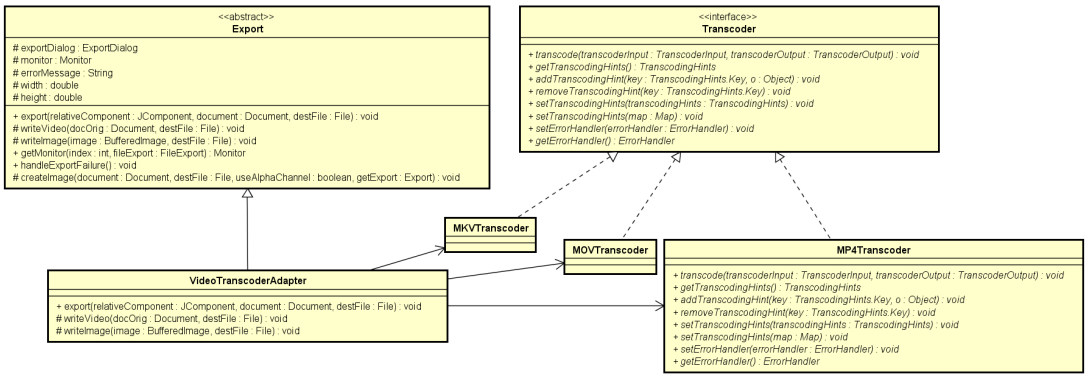
První parametr udává typ nastavení a druhý je jeho hodnota. Správu chyb zajišťuje rozhraní ErrorHandler, které obsahuje tři základní úrovně chyb – varování, chyba a fatální chyba. ErrorHandler lze přiřadit pomocí metody setErrorHandler.

Výše uvedené rozhraní bylo obsaženo již ve verzi Apache Batik 1.6, se kterou byl editor Glips Graffiti distribuován, bohužel vývojáři se jej nerozhodli použít. Implementace exportu do PDF a PNG využívají knihovny třetích stran. Pro PNG formát se jedná o PngEncoder (com.keypoint.PngEcoder). PngEncoder je třída, která převádí instanci java.awt.Image na pole typu byte, které představuje výsledný PNG soubor. PDF export využívá také knihovnu třetí strany a to Lowagie.

Kromě rozšíření o export do video streamu by bylo také velice výhodné mírně upravit architekturu exportního modulu. Aktuální stav není příliš rozšiřitelný a především obsahuje závislost na externích knihovnách. V případě exportu do video streamu se bez externí knihovny neobejdeme, nicméně export do PNG a PDF formátu lze velice snadno přepsat s použitím stávajícího řešení v knihovně Apache Batik. V zájmu budoucí rozšiřitelností by byla dobrá aplikace návrhového vzoru adaptér, který by sloužil jako rozhraní mezi abstraktní třídou Export a implementacemi pro export. Tento krok by přinesl znatelné zpřehlednění kódu a především snadnou rozšiřitelnost.

6.4.2 Návrh

K implementaci funkcionality využijeme modifikaci návrhového vzoru Adapter[6]. Díky této konstrukci bude zachována stávající struktura s využitím abstraktní třídy Export a zároveň bude využito rozhraní Transcoder, které je doporučené pro implementaci exportu SVG formátu, viz. Obrázek 23.



Obrázek 23: Export do video streamu

Využitím adaptéru je také umožněno snadné rozšíření o podporu dalších video formátů. Například můžeme přidat další třídu MKVTranscoder, která bude implementovat rozhraní Transcoder a ve VideoTranscoderAdapter se pak zvolí vhodná implementace.

Pro samotné řešení exportu do video streamu byla vybrána knihovna JCodec, která podporuje kodeky H264, MPEG 1, 2 a VP8.

6.4.3 Implementace

V první řadě byla implementována třída MP4Transcoder. Pro správu nastavení konverze je použita připravená třída TranscodingHints. Identifikátory pro nastavení výšky a šířky jsou převzaty z třídy ImageTranscoder. Pro potřeby video exportu byla přidána další konstanta, která reprezentuje snímkovací frekvenci. Typem je vnořená třída TranscodingHints.Key, která je inicializována její implementací BooleanKey.

Výchozí metoda, která spustí export, nejdříve zkontroluje uživatelské nastavení. Pro případ, že není nic nastaveno se použijí výchozí hodnoty (rozměr 640x480px a snímkovací frekvence 25). Následně se pokračuje inicializací vstupního dokumentu. V průběhu inicializace je nastavena instance třídy BridgeContext do dynamického módu, pak je spuštěn AnimationEngine a nastaven čas na 0ms. Tímto máme zajištěno, že veškeré objekty třídy TimedElement jsou správně inicializovány. Nyní se pokračuje cyklem, který iteruje po jednotlivých snímcích. Počet snímků je určen pomocí vynásobení snímkovací frekvence celkovou délkou animace. V každé iteraci je získán snímek BufferedImage, který je následně překódován na snímek ve formátu H264. Třída také obsahuje objekt Monitor, který určuje aktuální stav exportu, který je zobrazen v rozhraní.

Následně byla vytvořena třída VideoTranscoderAdapter, která implementuje rozhraní Export. Bázová abstraktní třída předepisuje implementaci metod export, writeVideo a writeImage. Metoda writeImage u videa postrádá smysl a tedy v jejím případě vracíme null. Metoda export zajišťuje vytvoření objektu třídy Monitor a jeho inicializaci. Po vytvoření monitoru se zobrazí modální okno s nastavením exportu. Potvrzením se zapíše nastavení a je zavolána metoda writeVideo, která inicializuje MP4Transcoder a spustí export. Zde je prostor pro další rozšíření programu. Dialogové okno s nastavením může nést mnohem více možností nastavení. Například,

zde může být výběr výsledného kontejneru. Uživatel by pak mohl zvolit mezi MP4 a MKV nebo MOV. Projevil by se potenciál výše zvoleného návrhu prostřednictvím Adapteru. Stačilo by pouze vytvořit další implementaci rozhraní Transcoder. V metodě writeVideo by se pak na základě výběru v dialogovém okně zvolila správná implementace exportu.

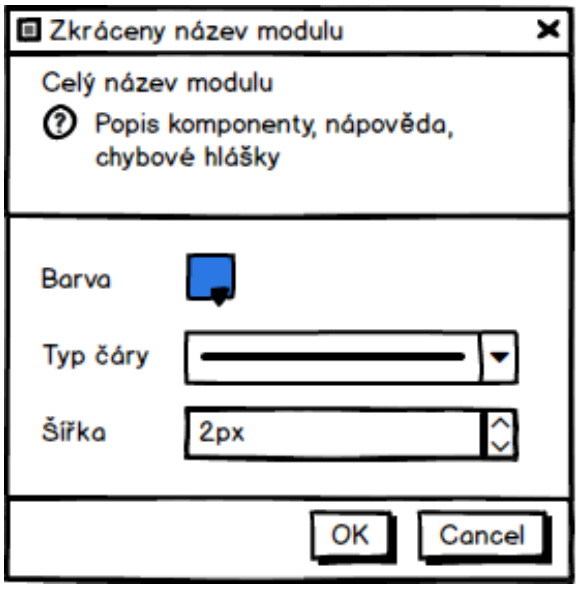
6.5 Ostatní rozšíření funkcionality

Kromě úkolů v zadání, byly do editoru přidány i další funkce, které usnadní práci. Jedná se především o zobrazení pomocných čar, které informují uživatele o tom, že daný tvar je animován. Čáry se budou vykreslovat podle typu animace. Například pro rotaci bude zobrazena šipka ve tvaru spirály, pohyb tvaru bude zobrazen křivkou, apod.

Dalším vylepšením bude implementace nového nástroje pro tvorbu rovných čar. Tato funkcionalita je v GLIPS Graffiti již částečně vytvořena, ale není uživatelsky přívětivá. Pero lze uzamknout pro kreslení čar, kde koncové body svírají úhel, který je omezen na kroky po 90°. Případně lze kreslit rovné čáry, ale uživatel si musí dát pozor, aby nechtěným pohybem nevytvořil křivku.

6.5.1 Zobrazení pomocných čar reprezentující animace

Pro zpřehlednění editoru byla naimplementována funkcionalita, která u animovaných elementů zobrazí pomocné čáry. Pro tyto účely byl vytvořen nový modul inspirovaný stávajícím, který zobrazuje mřížku. Nejdříve byl vytvořen návrh rozhraní pro nastavení pomocných čar, viz. Obrázek 24. Stejně jako u mřížky je možné nastavit barvu, typ a šířku čar. Nastavení se ukládá do registrů systému.



Obrázek 24: Návrh rozhraní pro správu animací

Modul je představován třídou AnimationsHelperModule, implementace je velmi podobná stávajícímu modulu GridModule, který byl popsán v kapitole 3.1.1 Moduly.

6.5.2 Nástroj pro tvorbu přímek

Jak bylo zmíněno výše, stávající nástroj pera není příliš vhodný na tvorbu rovný čar. Proto byla vytvořena upravená verze tohoto nástroje, tzv. zjednodušené pero, omezené na tvorbu přímek. Jeho implementace je velice snadná. Tento nástroj bude překrývat implementaci nástroje pera pro Bézierovu křivku. Vytvoříme tedy potomka třídy PathShape, který bude pojmenován jako SimplePathShape. Tato dědičnost nám předepisuje konstruktory předka, zde však není nezbytné nic měnit. Zavoláme původní verze pomocí klíčového slova super a přidáme do parametru identifikátor modulu. Pro zamezení definice křivky je nezbytné překrýt původní metodu notifyDrawingAction. Oproti původní implementaci odstraníme zachytávání události tažení myší, která zajišťuje deformování křivky. Tímto jsme zamezili standardnímu chování pera, kdy je tažením myši změněna pozice řídicích bodů. Nyní ještě je nutné vytvořit vlastní implementaci rozhraní IPathShapeHandler, které nám zajistí položku v menu a tlačítkové liště s napojením na třídu SimplePathShape. Samotná registrace modulu již pak probíhá přidáním záznamu do souboru modules.xml.

7 Návrhy na zlepšení

Možností jak dále vylepšit editor je mnoho. Příkladem může být nově implementovaný modul pro vytváření a vkládání vlastních tvarů. Stávající verze modulu umožňuje vytvářet vlastní tvary pouze za použití nástrojů tvorby cesty. Jistě vítaným rozšířením by mohla být podpora dalších tvarů, včetně těch, které jsou sloučeny do skupiny. Dalším vylepšením by mohla být podpora pro import a export. Momentálně se tyto tvary vkládají do registrů systému, což zajišťuje uložení pouze v rámci jednoho počítače.

Stávající řešení exportu do video streamu má poměrně strohé možnosti nastavení. Lze nastavit pouze rozlišení videa. Bylo by vhodné přidat podporu pro komplexnější nastavení, například úroveň komprese, kontejner, případně přidání různých filtrů (černobíle, sépie, apod.).

Modul, který umožňuje přidávání a editaci animací má momentálně předdefinovány pouze čtyři předpřipravené animace. Samozřejmě lze pomocí vkládání samostatných elementů vytvořit takřka jakoukoliv animaci, nicméně pro běžného uživatele je vhodnější mít více animací předdefinovaných.

8 Závěr

Cílem této práce bylo rozšířit vektorový editor GLIPS Graffiti. Aktuální verze tohoto editoru umožňuje vkládat pouze jednoduché předdefinované tvary. Prvním úkolem bylo přidání nového modulu, který umožní tvorbu nových tvarů, včetně jejich správy. Společně s tímto modulem bylo přidáno rozhraní, které umožňuje snadnou implementaci nových předdefinovaných tvarů. SVG formát, který GLIPS Graffiti primárně používá, umožňuje přidat animace k jednotlivým objektům. Tato funkcionalita byla do editoru také implementována. K tomuto účelu byl vytvořen modul, kde je možné přidávat animace k vybranému objektu. Každému objektu lze přiřadit neomezené množství animací. Pro uživatele jsou k dispozici jednoduché předdefinované animace a samostatné animační elementy, které umožňují vytvoření takřka libovolné animace. Výslednou scénu lze pak zobrazit v samostatném okně s časovou osou. V editoru byly rozšířeny také možnosti exportu o video stream ve formě souboru MP4.

Editor GLIPS Graffiti používá knihovnu Apache Batik. Tato knihovna pokrývá téměř všechny specifikace formátu SVG. Bohužel její dokumentace je velmi strohá. Pro vývojáře je k dispozici pouze vygenerovaná JavaDoc dokumentace, kde v mnoha případech scházejí komentáře popisující funkcionalitu metod. Vzhledem k rozsáhlosti knihovny a absenci literatury popisující tuto knihovnu bylo poměrně obtížné zanalyzovat animační možnosti knihovny a jejich využití v praxi. Výsledkem analýzy je popis základní práce s animacemi v této knihovně.

Přestože se výsledný editor nemůže rovnat komerčním řešením, je možné říct, že je poměrně ojedinělý. V době vniku této práce nebyl nalezen editor, který by měl podobné vlastnosti. Existuje mnoho open-source multiplatformních editorů, nicméně těch, které využívají SVG a podporují animace je velmi málo. Takový software je nepochybně velkým přínosem. Především z důvodu demonstrace animačních schopností knihovny Apache Batik.

Práce na editoru bude i nadále pokračovat. V průběhu implementace bylo nalezeno mnoho možností jak tento editor dále rozšířit. Prioritou je optimalizace kódu a publikování kódu na veřejný repozitář GitHub.

Použitá literatura

[1] Jaroslav Baláš. "Klientské rozhraní aplikací SCADA". Diplomová práce. 2006. URL: https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2007&a=balasj2&t=dipl.

[2] *Can I use*. URL: http://caniuse.com/#search=SVG (cit. 20. 04. 2016).

[3] *Co znamená SCADA/HMI?* URL: https://www.reliance.cz/cs/products/what-does-scada-hmi-mean (cit. 20. 04. 2016).

[4] I.F. Darwin. *Java: kuchařka programátora : [vzory a řešení pro vaše aplikace]*. Computer Press, 2006. ISBN: 9788025109441.

[5] Tomáš Fabián Eduard Sojka Martin Němec. *Matematické základy počítačové grafiky*. Vysoká škola báňská – Technická univerzita Ostrava a Západo-česká univerzita v Plzni, 2011.

[6] E. Gamma et al. *Návrh programů pomocí vzorů: stavební kameny objektově orientovaných programů*. Moderní programování. Grada Publishing, 2003. ISBN: 9788024703022.

[7] *GLIPS Graffiti Editor*. URL: http://glipssvgeditor.sourceforge.net/ (cit. 20. 04. 2016).

[8] Martin Gold. "Rozšíření Vektorového animátoru - grafická část". Bakalářská práce. 2013. URL: http://hdl.handle.net/10084/98797.

[9] Zdeněk Gold. "Rozšíření Vektorového animátoru - animační část". Bakalářská práce. 2013. URL: http://hdl.handle.net/10084/98935.

[10] *How to Set the Look and Feel*. URL: https://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html (cit. 20. 04. 2016).

[11] *Inkscape*. URL: https://inkscape.org/en/ (cit. 20. 04. 2016).

[12] Jakob Jenkov. *SVG Animation*. 2015. URL: http://tutorials.jenkov.com/svg/svg-animation.html (cit. 20. 04. 2016).

[13] *JFreeChart*. URL: http://www.jfree.org/jfreechart/ (cit. 20. 04. 2016).

[14] *KIYUT - Sketsa SVG Editor*. URL: http://www.kiyut.com/products/sketsa/ (cit. 20. 04. 2016).

[15] Alexander Kolesnikov. *Java Drawing with Apache Batik: A Tutorial*. BRAINY SOFTWARE, 11. břez. 2007. 249 stran. ISBN: 0975212893.

[16] Thierry Kormann. *Developing SVG Applications with Batik*. 2002. URL: http://www.svgopen.org/2002/papers/kormann__developing_svg_apps_with_batik/ (cit. 20. 04. 2016).

[17] S. McConnell a B. Kiszka. *Dokonalý kód: umění programování a techniky tvorby software*. Computer Press, 2005. ISBN: 9788025108499.

[18] *OpenOffice Draw*. URL: https://www.openoffice.org/product/draw.html (cit. 20. 04. 2016).

[19] R. Pecinovský. *Java 7: učebnice objektové architektury pro začátečníky*. Knihovna programátora. Grada, 2012. ISBN: 9788024736655.

[20] *Promotic*. URL: http://www.promotic.eu/cz/ (cit. 20. 04. 2016).

[21] *SCADA*. URL: https://cs.wikipedia.org/wiki/SCADA (cit. 20. 04. 2016).

[22] *SMIL – jazyk pro multimediální prezentace*. URL: https://www.interval.cz/clanky/smil-jazyk-pro-multimedialni-prezentace/ (cit. 20. 04. 2016).

[23] B. Spell a B. Kiszka. *Java: programujeme profesionálně*. Computer Press, 2002. ISBN: 9788072266678.

[24] *The Apache Batik Project*. URL: http://xmlgraphics.apache.org/batik/ (cit. 20. 04. 2016).

A Obsah CD

Fry0019.pdf Text této práce v PDF.

TechnickaPrirucka.pdf V tomto dokumentu je popsána struktura projektu a postup jak jej importovat do vývojových prostředí Eclipse, NetBeans IDE a IntelliJ IDEA. Dále jsou zde popsány doplňující diagramy z adresáře DoplnujiciDiagramy a testovací aplikace, která je zmíněna v kapitole 5.2 Základní práce s Apache Batik.

UzivatelskaPrirucka.pdf Uživatelská příručka popisuje základní práci s editorem GLIPS Graffiti včetně nové funkcionality, která byla implementována v rámci této práce.

DoplnujiciDigramy V tomto adresáři jsou vloženy diagramy, které z důvodu velikosti nebyly umístěny do textu této diplomové práce.

GlipsGraffiti.zip Archív obsahující spustitelnou verzi aplikace GLIPS Graffiti včetně nové funkcionality.

GlipsGraffitiSrc.zip Archív obsahující projekt aplikace GLIPS Graffiti.

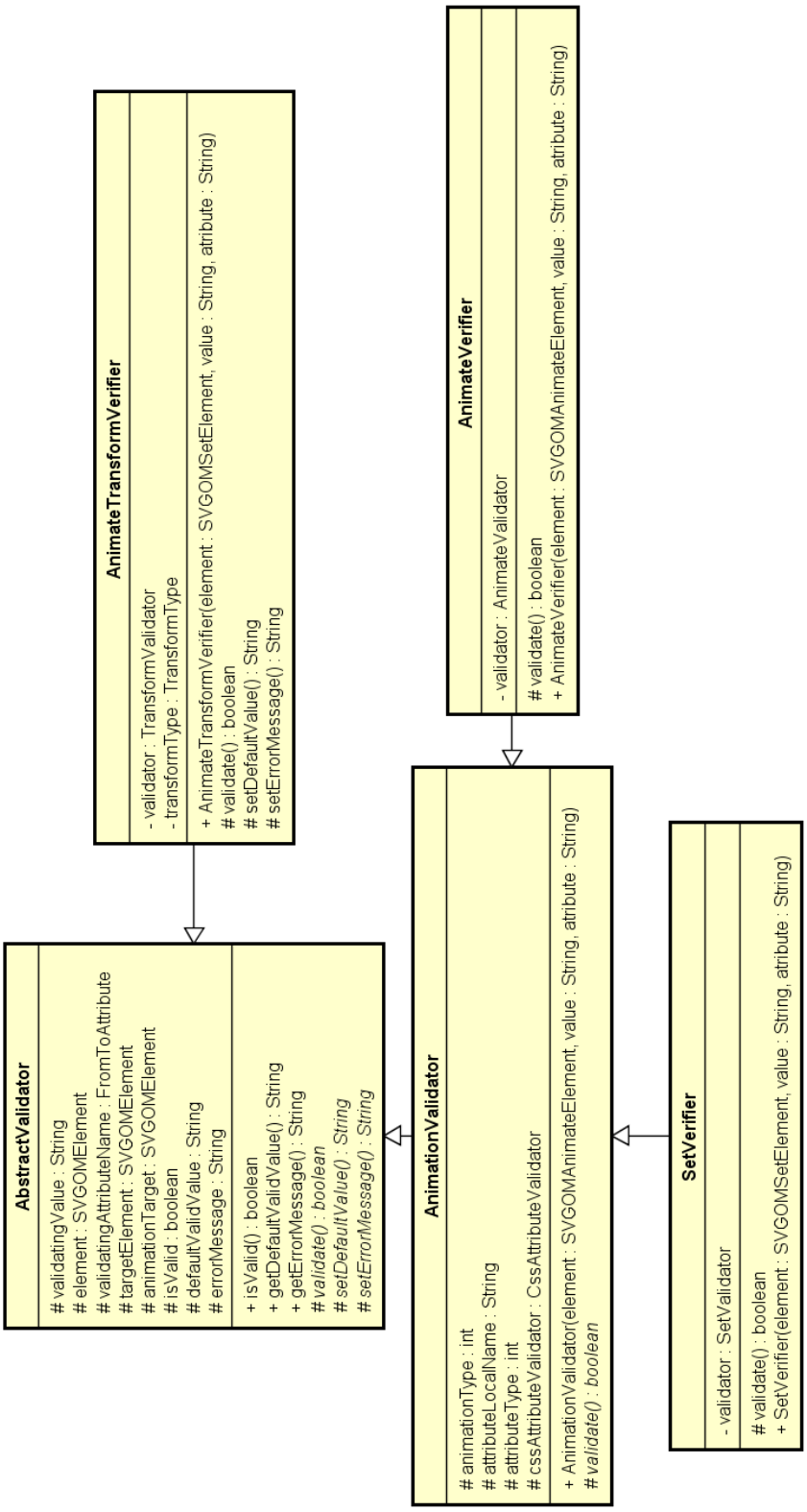
TestovaciAplikace.zip Zabalený projekt s testovací aplikací, která byla použita pro testování knihovny Apache Batik.

B Tovární třídy CSS a XML atributů

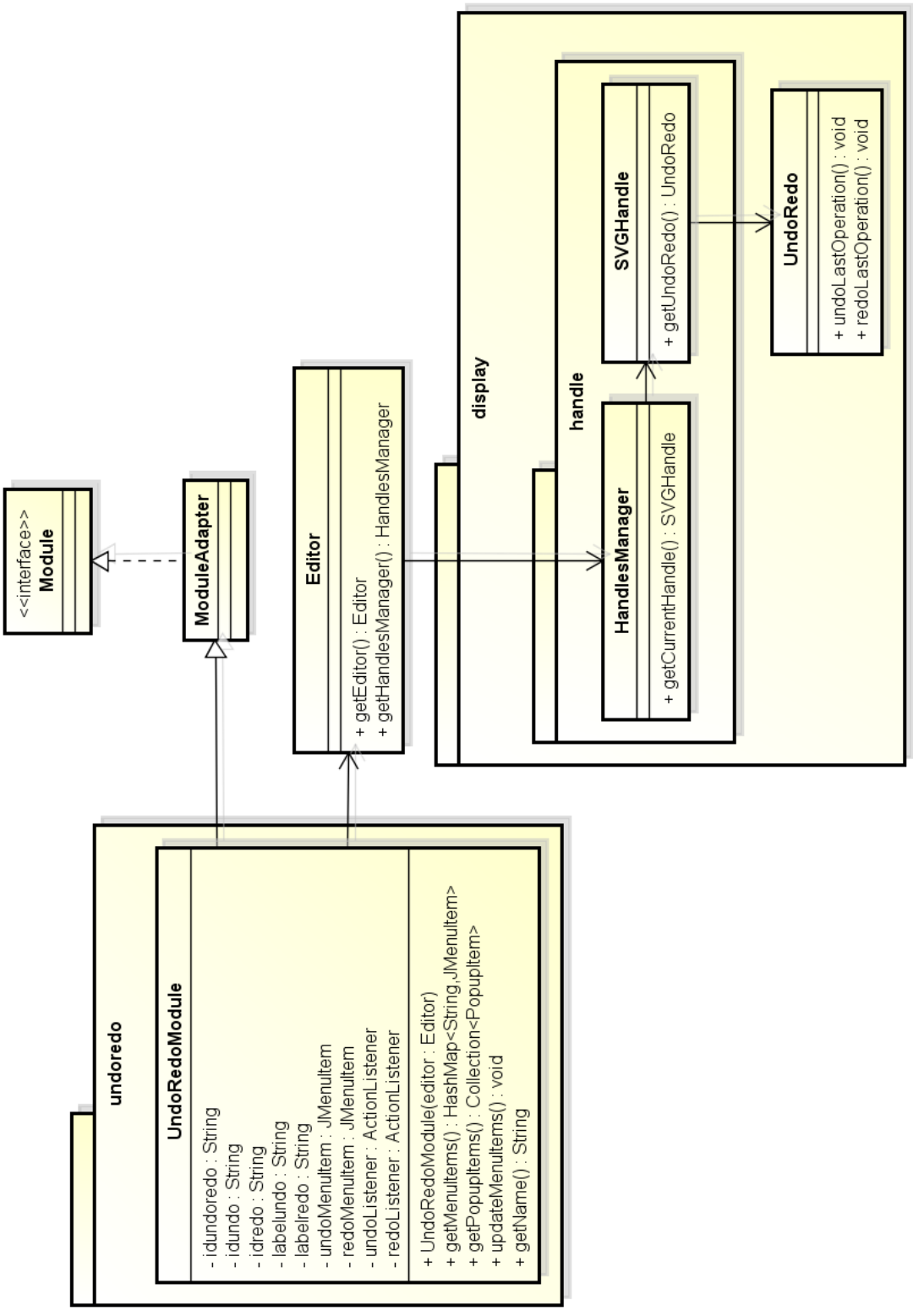
```
//Typ XML
org.apache.batik.bridge.SVGAnimationEngine$AnimatableIntegerValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableNumberValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableLengthValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableNumberListValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableLengthListValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatablePathDataFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatablePointListValueFactory
org.apache.batik.bridge.
    SVGAnimationEngine$AnimatablePreserveAspectRatioValueFactory
org.apache.batik.bridge.
    SVGAnimationEngine$AnimatableNumberOrPercentageValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableBooleanValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableRectValueFactory

//Typ CSS
org.apache.batik.bridge.SVGAnimationEngine$AnimatableLengthOrIdentFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableNumberOrIdentFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableAngleValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableAngleOrIdentFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatableColorValueFactory
org.apache.batik.bridge.SVGAnimationEngine$AnimatablePaintValueFactory
org.apache.batik.bridge.
    SVGAnimationEngine$UncomputedAnimatableStringValueFactory
```

Výpis 30: Seznam továrních tříd pro XML a CSS atributy

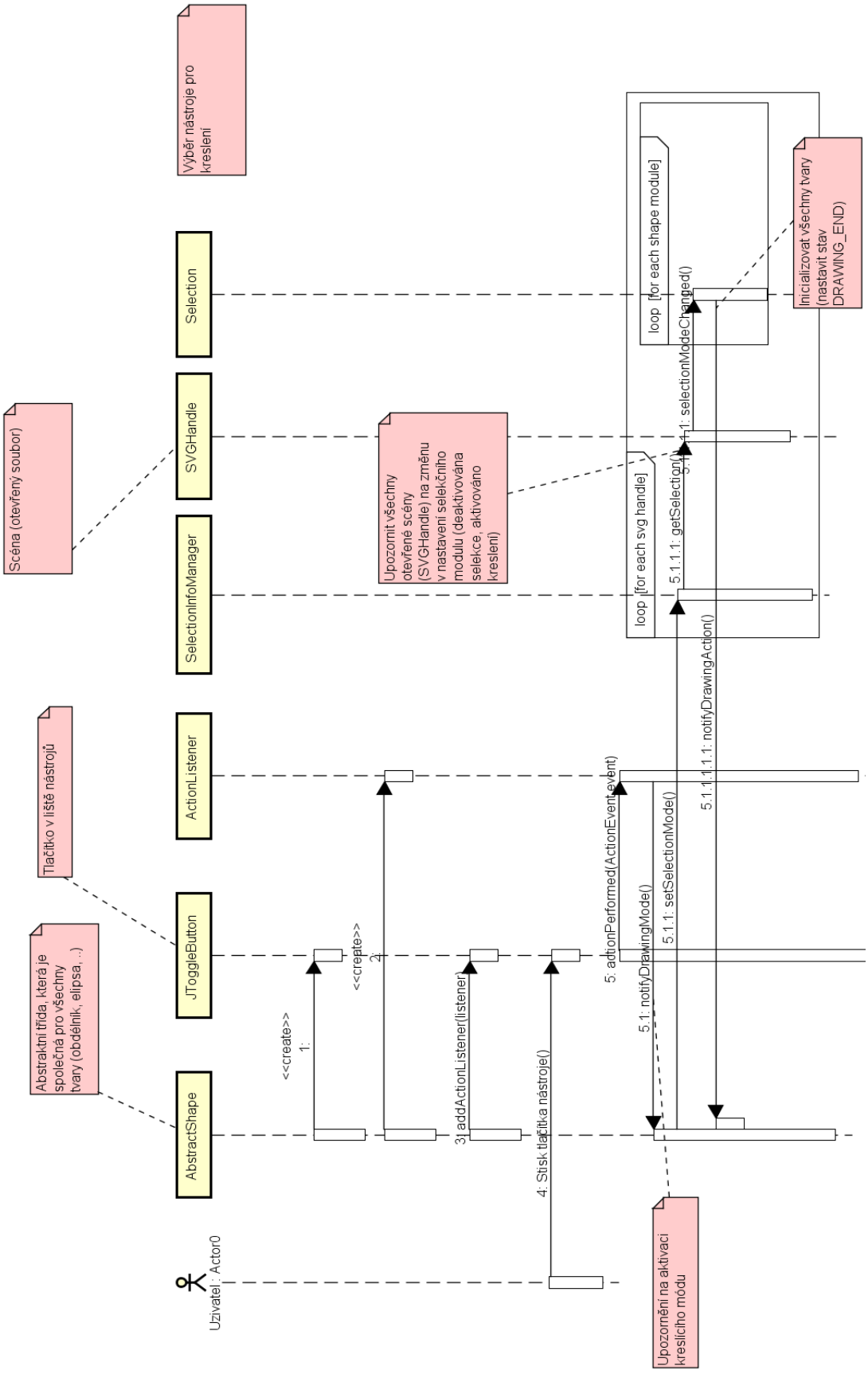
C Diagramy



Obrázek 25: Architektura validačních tříd



Obrázek 26: Modul UndoRedoModule



Obrázek 27: Sekvenční diagram popisující výběr nástroj

D Ukázky kódu

```
SVGHandle currentHandle = Editor.getEditor().getHandlesManager().
    getCurrentHandle();
        if (currentHandle != null) {

            Selection currentSelection = currentHandle.getSelection();

            if (currentSelection != null && currentSelection.getSelectedElements
                ().size() == 1) {
                Element selectedELement = currentSelection.getSelectedElements()
                    .iterator().next();
                if (selectedELement instanceof SVGOMPathElement) {
                    String shapeName = shapeNameEdit.getText();
                    if (shapeName != null && !shapeName.isEmpty() && !
                        currentCustomShapes.containsKey(shapeName)) {

                        SVGOMPathElement path = (SVGOMPathElement)
                            selectedELement;
                        String pathString = path.getAttribute("d");

                        //pokud cesta neni uzavrena
                        Path pathShape = new Path(pathString);
                        if (!pathString.contains("Z")) {
                            pathShape.closePath();
                            selectedELement.setAttribute("d", pathShape.toString
                                ());
                        }
                        PreferencesStore.setPreference(preferencesNodeID,
                            shapeName, pathShape.toString());
                        String colorString = Editor.getColorChooser().
                            getColorString(ColorManager.getCurrentColor());
                        selectedELement.setAttributeNS(null, "style", "fill:" +
                            colorString + ";stroke:#000000;");
                        fillShapesList();
                        setMessage(description, INFORMATION_TYPE);
                    } else {
                        setMessage(invalidNameWarning, WARNING_TYPE);
                    }
```

```
            } else {
                setMessage(noPathSelectedWarning, WARNING_TYPE);
            }
        } else {
            setMessage(noPathSelectedWarning, WARNING_TYPE);
        }
    }
```

Výpis 31: Uložení nového tvaru